**Title:**
A Unified Approach to Preserving Cultural Software Objects and their Development Histories

**Author:**
Kaltman, Eric, UC Santa Cruz
Wardrip-Fruin, Noah, UC Santa Cruz
Lowood, Henry, Stanford
Caldwell, Christy, UC Santa Cruz

**Publication Date:**
November 20, 2014

**Series:**
Graduate Research Projects

**Permalink:**
http://www.escholarship.org/uc/item/0wg4w6b9

**Keywords:**
digital media preservation, digital curation, computer games, video games

**Abstract:**
This white paper was made possible with the support of a National Endowment for the Humanities Digital Start-Up GrantHD-51719-13.

# A Unified Approach to Preserving Cultural Software Objects and Their Development Histories

## A Case Study in Academic Computer Games

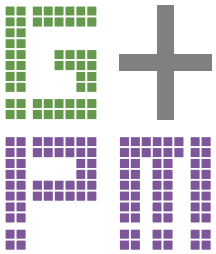
# A Unified Approach to Preserving Cultural Software Objects and Their Development Histories

## A Case Study in Academic Computer Games

Eric Kaltman, UC Santa Cruz
Noah Wardrip-Fruin, UC Santa Cruz
Henry Lowood, Stanford
Christy Caldwell, UC Santa Cruz

This white paper was made possible with the support of a National Endowment for the Humanities Digital Start-Up Grant HD-51719-13.

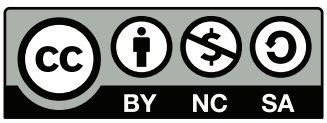
NATIONAL ENDOWMENT FOR THE
Humanities



GAMES AND PLAYABLE MEDIA
UC SANTA CRUZ

UC SANTA CRUZ UNIVERSITY LIBRARY

STANFORD UNIVERSITY LIBRARIES



## Acknowledgments

Many people helped with the creation and realization of this report. The authors would like to thank the core graduate development team of *Prom Week* (Michael Treanor, Ben Samuel, Aaron Reed, and Joshua McCoy) for their time and effort in creating and helping to organize the resources discussed in this report. The team was also courteous enough to allow interviews that greatly helped our understanding of the *Prom Week* development documentation. Thanks to *Prom Week* contributor Brandon Tearse and *Prom Week* advisors Michael Mateas and Noah Wardrip-Fruin for additional interviews. We would also like to thank Brenda Romero for raising project awareness and initial enthusiasm. Jess Waggoner and Susan Perry of UC Santa Cruz Library provided much needed help in navigating their digital repository. A thank you also to Jason Rhody of the NEH for support and guidance. And finally, a thanks to everyone, everywhere making and enjoying games (and other cultural software)!

# Contents

# Executive Summary

This project addresses a pressing problem for digital humanities, computing, and archival preservation. How can culturally significant software produced today be archived so that it will be available for scholarly research in the future? We address a specific and significant aspect of this general problem: software that is produced in universities and other non-commercial research institutions. In particular, we will focus on game software and its development. To date, little attention has been given to the problem of archiving academic software, and it is safe to say that virtually none has been given to academic game development. There are good reasons to address this problem. Academic data and software code is distinctive in many respects that we will illuminate in this study; these characteristics both enhance the value of academic software and explain much of the difficulty in its documentation and preservation. At the same time, in some cases the issues raised by game software and academic software also exemplify larger issues in the fields of software studies and game studies.

This summary provides an overview of the key findings in each section of the report, and a listing, in brief, of our recommendations and ideas for future research. The report is based on our case study of the game *Prom Week*, created by a team in the Expressive Intelligence Studio (EIS) in the School of Engineering at the University of California, Santa Cruz. *Prom Week* is a social simulation of the relationships between a group of high school students in the week before their senior prom. It is an academic research game that incorporates a new artificial intelligence framework, Comme il Faut (CiF), allowing the students in the game to remember past events and build unique and nuanced relationships. As both game and research software, we felt it could help in unpacking both the process of cultural software creation, and its resultant documentary traces. To help guide our initial investigations, we relied on previous work on the appraisal of science and technology records. Most helpful was the 1983 Joint Committee on the Archives of Science and Technology report, *Understanding Process as Progress: Documentation of the History of Post-War Science and Technology the United States*. It provided a basis for the appraisal of documentation resulting from scientific process, and argues – as we do – that preserving and understanding the documentation about how a project was conducted is as historically important as the final results. We share with this report a desire to elucidate the process of cultural software and computer game creation in hopes that future archives and cultural institutions will become better equipped to deal with such documentation.

## The Process of Academic Game Development

The first section of the report, "The Process of Academic Game Development", breaks academic game development down into six sequential processes:

### 1. Idea Formation

Most academic research projects are based on previous completed work, or develop along avenues aligned with the specific research interests of a laboratory or research group. *Prom Week* is the result of an accumulation of other research and system development work that represent years of previous efforts in the fields of artificial intelligence, sociology, and game design. Because of the varied paths that some research projects take, new research sometimes begins without a distinct awareness of its new research direction. As a result, early project documentation is fundamentally tied to previous areas of study, and any researcher or archivist hoping to deal with documentation of academic game development should be aware of the potentially deep connection any project has with its predecessors.

### 2. Physical Prototyping

After the idea for a game develops, the designers may want to test out various parts or a total approximation of the game system. Physical prototypes are stripped down, physical analogs of a game system intended to save time by avoiding unnecessary programming and other complex computational tasks. Prototypes provide a first look at a game's player interaction mechanics, and the feedback loops created through play. Ideas that seemed great on paper may confront obstacles or irreconcilable problems in this stage and need to be re-thought and redesigned. For *Prom Week* specifically, an early prototype provided feedback that changed crucial aspects of the game's design, and also caused significant changes to the underlying CiF architecture. In this way, the prototype also influenced the scientific focus and development of the project.

### 3. Digital Prototyping

When physical prototyping is concluded, or in some cases deemed infeasible, digital prototypes are constructed as proof-of-concept. Many software systems, including games, are inherently complex computational objects and it may only be possible to understand their specific architectural and technical challenges through probative creation. Digital prototypes also represent initial investigations into the development platform and environment used for a project. Development platforms are the specific hardware and software requirements needed to run a piece of software, and a development environment is all the tools and other developer-focused items that support software creation for a specific development platform. Digital prototypes also introduce a variety of born-digital documentation that will accompany development from that point forward. In certain academic contexts, a digital prototype or demonstration program might be all that is needed for a specific research goal.

### 4. Development Methods

If a project is going to be the focus of a sustained development effort it will employ some type of development methodology to manage team communication, feature creation, and other development tasks. Two popular methods for game and software creation are the waterfall and iterative strategies. Waterfall development begins with a comprehensive design document and strict scheduling of project phases and tasks. In this way, each development phases cascades into the other, like a waterfall. This development method is tied to specific scheduling and deliverables and is generally less flexible in the face of unforeseen issues and developments. Iterative development focuses on creating less potentially unnecessary work by organizing tasks and development effort into shorter cycles of consistent improvement. Scheduling in this strategy is less strict, and the onus is placed on the completion of specific tasks that might end up changing or become more constrained as development continues. Iterative development tries to complete a certain minimum amount of functionality before committing to greater depth or complexity. In reality, both methods turn out to be messier and more chaotic. In academic games and research, development strategies also run concurrently with scientific publication and changes in research direction. *Prom Week*'s development progress is mirrored in publications introducing and relating different parts of the game's systems and design. In this way, academic game development strategies and academic research are entwined throughout the duration of a project. An understanding of the documentary outputs, constraints, and process of both development and research is necessary to understanding the larger academic game project.

**5. Release and Dissemination**

Academic games produce both a playable experience and a set of academic publications. If the game is intended for a larger audience, it needs to be released in a shareable format, either on-line or through physical media. Additionally, any project innovations or novel developments are disseminated through peer-reviewed journals and academic, software, or game-related conferences. An institution may want to promote specific research publications or the game itself, involving a whole host of complementary public relations and press documentation. Release is also one of the most dangerous times for game documentation, especially if development is scheduled to conclude upon release. It is imperative that arrangements are made for the documentary output of the development and research process to be preserved in addition to the final outputs of publication and software.

**6. Refactoring and Continued Development**

In some cases, release signals the end of development, and in others the beginning of prolonged maintenance and improvement. Some projects rely on data generated by a software's release, and will continue generating research documentation and data for a significant period of time. This aspect of project is potentially problematic for any preservation effort, since there is no clear means for continuous retention of data. Large scientific projects do maintain and follow data management plans for continuous experiments, but if there is not a concerted effort by a development or research team to allow for long term preservation of data streams and/or continuous software updates, much of this subsequent work will be lost.

## Project Organization in Context

The second section of the report, "Project Organization in Context," describes the internal and external relationships present in an academic laboratory that creates games and other cultural software. In this case, we focused on the EIS lab, in the Department of Computer Science at UC Santa Cruz. Essentially, laboratories need to navigate internal relationships with the department and larger university above, and with the researchers and students below. An academic research lab is usually founded by a faculty director with a specific research plan. The director then takes on interested undergraduate and graduate students to help further that plan or complete goals. A significant amount of the documentary and research output from an academic research lab will come from, and be managed by, student researchers and assistants. Many times graduate students will be tied to specific grant funding and co-author papers with the lab's director. In the case of *Prom Week*'s development, both of the EIS labs co-directors had a hand in its development direction and foundation. Graduate and undergraduate student developers were responsible for most of *Prom Week*'s documentary, research, and software outputs.

Academic labs must also interact with outside constituents, and will often modify their own research direction and scope in relationship to the work of others in the community. For academic game development work, interaction with other researchers, both outside the department and even the particular school, are often necessary due to the interdisciplinary nature of computer games. Many games involve significant artist and narrative effort, and a specific subjective focus or theme that might require the involvement of humanities researchers or other disciplinary experts in non-Computer Science related fields.

## Document Methods and Types

The third and final section of the report, "Document Methods and Types", is an overview and appraisal of the significance of all relevant development documentation collected from our study of *Prom Week*. Most of the documentary classes and digital content types will be similar or related to other academic game or cultural software projects. Broadly, the section describes the following documentary types:

### *Physical Content*

The development process includes the potential creation of physical prototypes, press materials, and other promotional material (t-shirts, stickers, etc.). This content needs to be included in any comprehensive preservation or appraisal strategy.

### *Game Development Files*

Game development files are those born-digital files including:

1. **Software development** files associated with a game's technical implementation.
2. **Active research** files associated with academic publications, commentary on research results, research data, or themselves secondary research software.
3. **Self-documentation** files created by the development team to document the game and its process for press, conference presentation and collaborative development.

Software development files include source code, creative content (any audio or visual assets), and external assets (files not created by the development but used in the creation of the software). External assets may include licensed software libraries or tools used in development, and may be subject to legal restrictions on storage and reuse.

### *Prom Week Files*

Our study of *Prom Week* revealed over 60 different file types associated with over 30 different third-party programs. This represents a pressing future issue, in that many file formats are proprietary and not sufficiently documented to the extent needed for long-term preservation efforts. Even different versions of the same file type may present future compatibility issues. The obscurity of proprietary file types is also an issue that will only grow, as more and more generations create and abandon software. Some of the file types in *Prom Week* are already at risk of future loss.

### *Cloud Services*

Many of the organizational documentation, creative content, and research files for *Prom Week* were stored on various cloud services, including Google Drive and Dropbox. These services are not designed for long-term preservation of data and are at risk, both by the whims of the commercial marketplace, and by their design. Preservation of provenance information, including creation, file ownership, and file modification dates, are paramount to the historical record of a development process. Both Dropbox and Google Drive provide facilities for the preservation of this information, but they either require additional fees or an understanding of Application Programming Interfaces (APIs) to connect to underlying web services. Cloud services also have access restrictions to content, and thus required us to communicate with the development team to enable migration of data off of the services, which might not be an option for projects not archived at their home institution.

### *Version Control*

Version control systems manage file changes and revisions for a software development team. Most projects involving multiple programmers, artists or other contributors, make use of version control to streamline and manage development files. These systems also contain

most of the data necessary to compile and run the software being developed. It is important to note that the version control systems themselves are also subject to the same preservation issues that haunt all born-digital files. Therefore, it is important to save not only the files stored in version control but also the version control configuration and organization files.

### *Repository Storage and Preservation Strategy*

The third section concludes with a discussion of the efforts to store *Prom Week* in the University of California's Merritt digital repository, and our concurrent efforts to organize and preserve the output of undergraduate game development classes in the School of Engineering at UC Santa Cruz.

## Recommendations

Below, briefly, are the general recommendations of this report Our complete recommendations to cultural and academic institutions dealing with cultural software development files are included, in depth, at the conclusion of this report.

1. Establish a laboratory- or department-wide data management policy that outlines how data will be collected, organized, described and archived, according to the standards of the intended institutional or discipline-specific data repository.
2. Produce data and documentation in formats that are easily archivable or can be converted to easily archivable formats.
3. Store all relevant development documentation in as few separate locations as possible. Create a manifest or directory that describes and provides access to all project documentation.
4. If possible, ensure that your source repository is available via the open Internet in read-only form.
5. Provide source control check in comments that are descriptive and can be understood by repository managers who were not involved in your project.
6. Record software programs used in the creation of files and assets (including format and version information).
7. Conduct development correspondence on official group mailing lists as much as possible.
8. Assign responsibility for digital archives and institutional software development archives to a specific member on the document appraisal team. Ensure that this team member is well-versed in software development processes and documentation.
9. Instate data management plans for all game design and development projects and courses.
10. Develop instruments (such as transfer and data deposit agreements) and policies for ingestion, description, and access with regard to institutional software development records.

It is our hope that in reading through this work, one will be made aware of the breadth of development documentation, and the need to preserve not just the final outputs of scientific research and software development work, but also the documentation of processes, false-starts, and failures. Much of the history of cultural software is already lost or at risk unless we find organized, methodical, and sensible means to ensure its long-term preservation. This report is an initial step toward further research and awareness of this type of process documentation.

# Introduction

This project addresses a pressing problem for digital humanities, computing, and archival preservation. How can culturally significant software produced today be archived so that it will be available for scholarly research in the future? We address a specific and significant aspect of this general problem: software that is produced in universities and other non-commercial research institutions. In particular, we will focus on game software and its development. To date, little attention has been given to the problem of archiving academic software, and it is safe to say that virtually none has been given to academic game development. There are good reasons to address this problem. Academic data and software code is distinctive in many respects that we will illuminate in this study; these characteristics both enhance the value of academic software and explain much of the difficulty in its documentation and preservation, as we will show. At the same time, in some cases the issues raised by game software and academic software also exemplify larger issues in the field, as we will also discuss.

## OVERVIEW

Most cultural expression today is created with, disseminated through, composed of, or mediated by computers and digital technology. Artistic and cultural works frequently depend on the machinations of software. This is why it is imperative that cultural institutions devote attention to software archiving and preservation. An increasingly important expressive use of software is the creation of interactive computer games. Games not only occupy a commercially successful sector in the media landscape, they are also growing in cultural importance. Digital humanists, game studies scholars and designers are paying attention to digital games – both commercial and non-commercial.

We believe that most libraries and archives, the institutions that will be challenged with the task of preserving archives and collections of game software development for future scholars, have not yet fully grasped the nature of computer games either as a medium or as a created artifact, and thus, as a collection object. An important reason for this knowledge gap is the lack of procedural guidance in handling, appraisal and retention of complex computational objects. Our primary goal is to present a preliminary framework for understanding the different facets of the processes that create games in terms that will translate into effective institutional archives of academic software development. Future scholars and designers will need well-organized and viable archives to understand the history of cultural software objects. This report lays the foundation for building software archives on a sound understanding of software development.

### *Prom Week*

We will focus on the development process of *Prom Week*, a social simulation game created by the Expressive Intelligence Studio (EIS) at UC Santa Cruz. We chose it for a variety of reasons. First, it is a complex, multimedia object, as most games are, and its creation required significant multidisciplinary effort. A team of programmers, game designers, artists and authors collaborated in realizing its many design goals. Second, *Prom Week* provides an excellent test case for archival treatment of software development documentation. Members of the creative team were also active computer science researchers. *Prom Week* is both a full-fledged computer game and an academic research project. Institutional responsibility for the preservation of scientific research data and documentation has in recent years become a hotly discussed topic for digital repositories. Therefore we can leverage insights from work on scientific data management and preservation and apply methodologies developed for those purposes to archives of cultural software such as games. Lastly, *Prom Week*'s development at a public university allowed for unrestricted access to documentation of all development processes. Few accessions of software documentation will include as complete a collection of all aspects of a development team's work. Our treatment of *Prom Week* is therefore an exhaustive case study through which we attempt to address as many types of documentation as possible. This white paper will use *Prom Week*'s development process and our access to it, as an illustrative proxy for guiding future archivists through their own software development collections.

### *JCAST Report*

A crucial framework for our investigation is the 1983 Joint Committee on Archives of Science and Technology (JCAST) report Understanding Process as Progress: Documentation of the History of Post-War Science and Technology in the United States. It is an argument for institutions and archivists to gain an understanding of scientific and technological development processes as way to better understand the history of scientific progress. The report highlights three major problems in dealing with historical scientific data and records. These problems map, without much translation, to major issues in digital computer software appraisal and preservation.

1. The amount of unpublished documentation in academic game development is not addressed (or categorized by) current archival practice, and it cannot be estimated based on experiences from other fields.
2. There is "an absence of professional consensus on guidelines for the appraisal and description of archival records of science and technology." This lack of consensus contributes to both ingest backlogs at repositories unversed in the material and, in some cases, might lead to the needless destruction of potentially valuable materials. Many institutions are unaware of what they have, what can be done with it, who would want it and what it is worth.
3. Too little is known about the potential users of academic game documentation or "about how adequately contemporary [archival] practices [meet] their needs."

This report begins to address these problems and provides recommendations to institutions and archives dealing with cultural software documentation. We discuss the process of software creation, both in terms of academic context and documentary output, in order to highlight issues inherent in the appraisal and preservation of the archival records it leaves behind. Our discussion focuses on common steps in game development and computer science research as process. *Prom Week* is both a research object, in its contributions to social modeling in artificial intelligence systems, and an independent game that received significant press attention from the gaming community. Therefore, our notion of process includes both scientific research and game development, and both are discussed in our analysis.

## Development Process

The academic game development process is broken down into six sections:

1. Idea Formation
2. Physical Prototyping
3. Digital Prototyping
4. Development Methods
5. Release and Dissemination
6. Continued Development and Study

Each section presents a phase of the software project's process, with a complementary discussion of documentation types created during that phase. There are no specific recommendations concerning which documents to preserve. We hope that in illuminating the document types and development processes, archivists will be able to develop appraisal strategies that narrow down what is important for a specific project. No two development processes are the same; however, we believe that they usually follow the structure above. This structure is based historically on the Charles Babbage Institute's (CBI) 1989 report on the Control Data Corporation's records.[1] In that report, different business processes were separated and explored through the use of targeted case studies concerned with specific company products, like the CDC 1604 computer. All documentation relating to each product was located and reviewed to gauge the extent and diversity of the entire organization's records. This "documentary probe" allowed the researchers to organically discover the business's internal processes through the examination of documentation and, in turn, cast light on the types and roles of documentation created by those processes. The *Prom Week* case study will provide a similar framing mechanism for academic research and game

1 (Bruemmer and Hochheiser, 1989). See bibliography for full citation.

development as a set of processes and a collection of data and documentation suitable for appraisal and storage.

### *Process in Context*

The process sections are followed by a discussion of archival records in an academic context. As discussed in the JCAST report and other similar inquiries into academic laboratory organization and politics, the processes of discovery and development cannot be divorced from the social organization of the people conducting the work. An overview of the organizational structure of a laboratory highlights the different perspectives and goal structures possessed by individual researchers. The report's second section describes the different roles in an academic game research lab. The distractions and commitments of researchers are addressed. In addition to discussions of lab organization and personnel, there are subsections devoted to the interface between the lab and outside organizations. The EIS lab has ties to the Computer Science Department, other UCSC departments, game research as a field, and professional game development communities. Interactions among these groups are described below. All documentary production is embedded in the social processes at work in a lab, and it is imperative that archivists understand the social commitments and motivations underpinning academic game development.

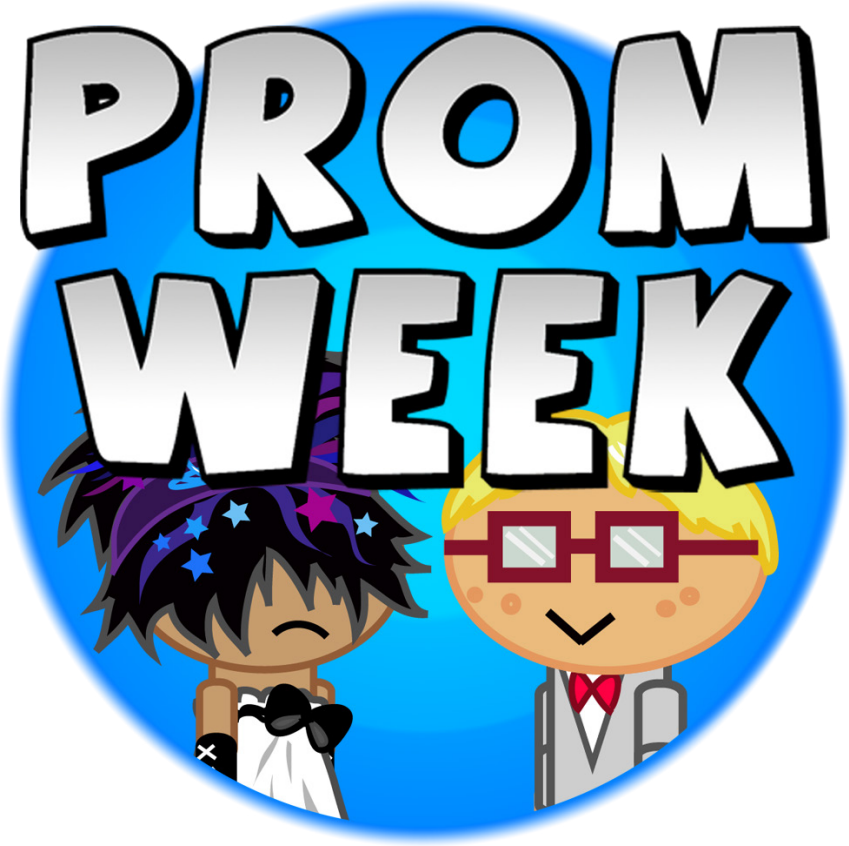
### *Documentation*

The third section is devoted to discussion of the diverse documentation created by development processes. We examine documentation outlined in the process section in more detail here. The reason for the separation of process and documentation is that the range of records encountered through the probe of *Prom Week* requires understanding of networked documentation, cloud services, and technical descriptions of various game design and software files. This section will highlight the channels (cloud services, notebooks, etc.) through which software development documentation circulates. It will provide a starting point for alleviating the confusion associated with diving into software development documentation for the first time. The goal is to provide an introductory framework for the archival appraisal of software development documentation and for examining the options for making these documents available to future researchers. Scholars, game designers and students may be interested in different types of documents and it's important that each kind of record (whether physical or digital) is given some fixed position in an archival appraisal and retention strategy. Each documentary discussion is paired with the real complexity of *Prom Week*'s documentation. We address different methods used in locating and finding *Prom Week* documentation, and some basic methods for analyzing development files.

### *Recommendations and Future Research*

The paper concludes with summary recommendations gleaned from the preceding exegesis along with a discussion of future research directions and potential.

# The Process of Academic Game Development




## OVERVIEW

Game development and scientific practices are both highly specialized, incorporating terminology and epistemological frames that create a seemingly impermeable membrane between the internal processes of a discipline and their perception by those outside the community. This barrier prevents archivists, librarians, and information professionals from understanding how to evaluate software projects and programs. Part of the reason for the impenetrability of software is its primary function as a commercial product. Companies want to protect their work from illegal duplication and piracy, and therefore intentionally hide or obfuscate access to their code, development practices and internal tools. While this concern might be valid in the accelerated world of commercial software, the implications for archives are a distinct lack of knowledge about how development processes function and a dearth of understandable documentation as to how one might archive and retain such work. The ability to retain, reproduce, and understand a software object is increased when more information about it is available. Another difficulty is a lack of general explanation about how development processes function and produce documentation. Many software projects produce an enormous amount of digital files and data. Parsing this barrage of information is beyond the abilities of all but the most seasoned developers or specialists. Our goal is to help alleviate some of the confusion around development processes and documentation, and to provide recommendations for a more informed treatment of software by libraries and archives. Access and understanding are needed for any software preservation effort to thrive. The choice of *Prom Week* as a case study for new recommendations and guidelines in software documentation appraisal and description stemmed from our access to its documentation and our familiarity with its development processes. Our analysis of *Prom Week* grounds our discussion of general software development issues and allows for a more nuanced documentary understanding of development processes.

The ease of access to *Prom Week*'s documentation stands in contrast to normal game development practices. Commercial game companies are some of the most secretive

development outfits in the entire software industry. Most craft knowledge of advanced game design and technical architecture is closely held within individual development studios. The game as product is the major motivation for the time and effort commercial studios put into their work. The extensive development process is thus encapsulated in an executable file and released. No source code or technical documentation is usually available.

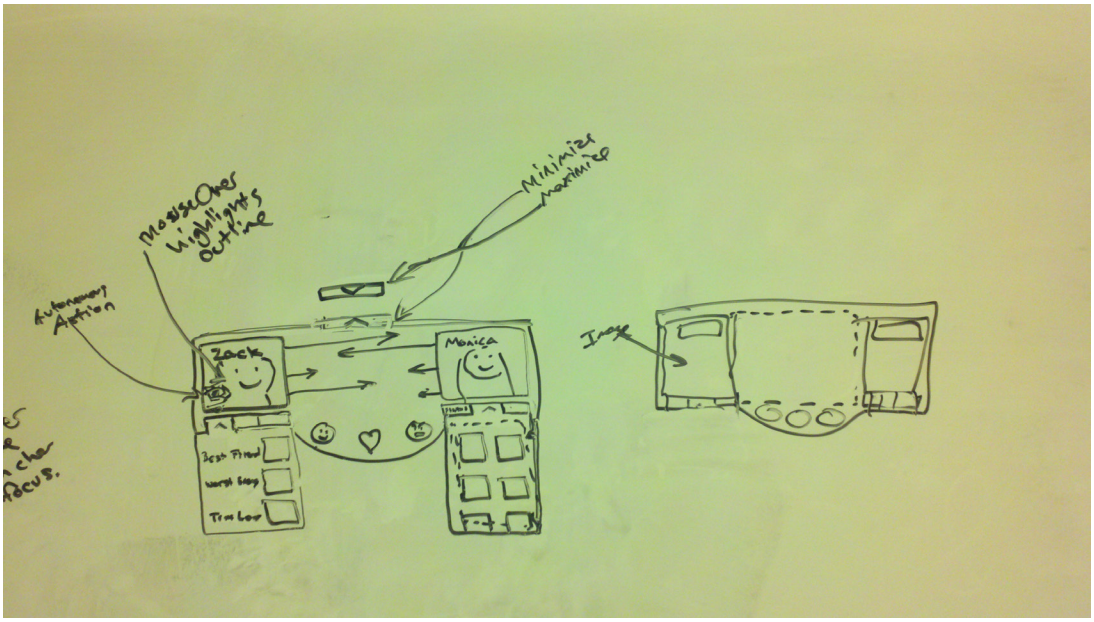
On the contrary, academic and scientific investigations focus their output on the production of papers, articles, conference presentations and monographs. These outputs describe system creation and implementation in varying levels of detail, but the main motivation is to share the work and technical effort with colleagues to get feedback and further academic careers. In many cases grant funding will require a data management plan for anything produced by a project. Access to academic development documentation is therefore easier to obtain but may still be disorganized, incomplete or not well documented. Additionally, most scientific research process (the reality of scientific research) is hidden behind the publications resulting from it. The main impetus for the JCAST report, a methodological source for this white paper, was that scientific processes and their documentary records were hidden and obscured by their publication outputs. Only "correct" and efficient findings, not failures, are presented and accepted for publication, hiding much of the mess and actual scientific work pursued in research institutions. For future scholars, scientists and historians, understanding how we create knowledge is just as important as what that knowledge asserts. Most scientific progress is based on reproduction and extension of others' work, making that work easier to access and more transparent would open up more research avenues and increase scientific output. Software development research, in particular, would benefit prodigiously from better access to early systems, source code and documentation. *Prom Week*'s academic development context allowed us to conduct document collection and analysis relatively uninhibited. Our familiarity with the *Prom Week* development team (two of our investigators are members of the same research group) also helped to surmount the opacity of the game's development and research process. These personal connections and familiarity with lab procedures and routines greatly expedited documentation access to the development process. The more open a platform is, the better a chance it has for migration to other systems, and the more open a development process, the easier it will be for future researchers and students to understand and further the work.

*Prom Week*'s position at the nexus of professional and academic software design methods made it a unique software object, and allowed us to better connect its documentary production with other game development efforts. *Prom Week* took over three years to create and led to numerous innovations in the field of artificial intelligence, mainly in social modeling systems for games. Additionally, *Prom Week* received numerous accolades and positive attention from the general videogame and technology press. *Prom Week* is a rather large academic project with over twenty individual contributors; however, we were still able to cover most of the documentary traces related to the project. The project was also recent enough that most everyone involved still had active knowledge of its development.

Abstracting general processes from a singular case is inevitably partial, and in the worst cases misleading. We are careful in the following discussion of development process to note that not all the events and milestones present in the development of *Prom Week* are generalizable. However, we are confident that the shape of the documentation, the practices used by the developers (many of whom have professional development experience), and the development timeline are fairly consistent with modern game development practices for small teams. Previous documentary probes following JCAST recommendations expanded documentary understanding of scientific, technical and business practices. This study of *Prom Week* seeks to

do the same for small-scale software development work. Games are software, so the following explanations and recommendations will be applicable to all collections of software-based works.

Each major developmental stage will be highlighted below. A general description of each stage is punctuated with examples from the *Prom Week* case study and followed by a description of the types of documentation produced at that point in the development process. Descriptions are guided by the investigators' experiences documenting *Prom Week*, as well as their own knowledge of game development – having also developed and archived games, as well as taught game design, at research universities. The documentary discussions will try to reveal the ontology of game design and game development records, illuminating the various types of documentation important to the overall development process.




Early User Interface Prototype Sketches

## Idea Formation

In the process of scientific and technological innovation there are rarely flashy moments of insight and the immediate creation of new paradigms. Seemingly instantaneous discoveries are the result of longer, uncelebrated periods of hard work and significant deliberation. In both scientific work and software development only the final outputs of a prolonged, active investigation are revealed to those outside the immediate laboratory group or invisible college.1 The work of academic game development and study occupies two different technical traditions. In one case scientific papers are the desired output, furthering the disciplines of computer science, game studies, game design and many others. In the other, a software artifact, to be used and abused by a specific audience, is the goal of the many hours of technical labor and design. These results and the processes behind them are tied together in the development of academic games. Academic games are produced for a variety of reasons, for example: to reify theories about computational system or game design; to collect data to support a social or user-based hypothesis; or to crowd source solutions to complex, distributed problems. Our specific focus is on *Prom Week* and our conclusions are based on its goals as a theoretical object. *Prom Week* is a proposal for a new type of AI system and a data-gathering tool for how players interact with that system. As such, our presentation of academic development and research is tinted by *Prom Week*'s specific methodological lens. This section will discuss the means through which academic game design ideas form and how the processes of scientific inquiry inform their design.

Academic games generally fall into two different ontological roles; they either are the object of study or help further the goals of a separate scientific investigation. Appraisal and retention of the tools developed by scientific research is covered in the JCAST report, but only in a very cursory light. It is assumed that the tools themselves are secondary constructs to the actual work of scientific discovery. Therefore it is recommended that tools used for discoveries should only be saved in very specific circumstances. Research in game design studies and computer science is different. Generally, the tools of computer science research are central to the process of knowledge creation. This report affirms the importance of games and related support software and structure as the point of the

1 Researchers in the same lab or at complementary labs at other institutions specializing in a similar topic.

research and the major axis of innovation. *Prom Week* is a contribution to the fields of AI and narrative game research and that contribution comes from its existence as a playable object and system. Tools and implementation details are necessary to understand *Prom Week*'s specific contributions to relevant disciplinary fields. Understanding game software and its development tools is necessary to comprehending its historical position in AI research.

Academic games created as scientific arguments follow a familiar research trajectory. A group of researchers, generally faculty advisors and graduate students, leverage previous experience and work into the study of a new technical innovation or idea. This process is similar to the stages of the basic research model: hypothesis, experimentation and testing, and results. The design of any new computational system or game is based on certain epistemological assumptions grounded in previous research. Academic researchers build on previous work and the curiosity of lab members to organize different directions for research. They project forward, tuning lab research to potentially fruitful areas of study. These inquiries are codified into grants or other official research proposals if the direction is (somewhat) clear. Other times avenues of research are pursued on the side of official funding channels, for instance in classwork, during personal time, or as the result of unexpected research outcomes. The process of idea generation is usually clear in retrospect, so the purpose here is to highlight that many internal paths lead to promising research. Certain documentation may seem trivially important at one point but upon reflection may constitute the foundations of a larger, more coherent research goal.

### *Prom Week Genesis*

In the case of *Prom Week*, the genesis of the game's design and creative spark begins with the Expressive Intelligence Studio (EIS), a computer science lab largely devoted to artificial intelligence for the creation of novel media experiences at the University of California, Santa Cruz. Early work in interactive narrative systems by the lab's co-founder, Michael Mateas, drew interested graduate students who wanted to help develop novel and innovative systems for story creation and interactive, dramatic computer games. One graduate student, Joshua McCoy, developed the initial prototypes that lead to *Prom Week*'s creation. McCoy had degrees in sociology and computer science and combined his interests, modeling character interactions while drawing on various sociological and dramatic theories. This work, specifically inspired by Erving Goffman's dramaturgical analysis, which theorizes interactions between people through the metaphor of a stage play in which individuals "portray" themselves in different ways in different social contexts, lead to investigations into the social roles functioning in fictional works like film and television dramas. Research continued with the integration of theories in motivational psychology and sociology into the design of a computational system for imitating dynamic social interactions among various computer controlled agents. The express goal of the initial research into *Prom Week* is thus a combination of the lab's guiding vision and the initiative of its leaders and graduate students.

After significant preparatory effort, a decision was made to try and develop a complete computer game based on McCoy's social system research and years of related work by the lab's founders into the computational modeling of narrative interactions. This project would eventually become *Prom Week*, but it was the result of years of previous work that can only be historicized retrospectively. Scientific research in computer science labs is the result of many different influences, in this case the outgrowth of previous, self-motivated inquiry into an interdisciplinary combination of sociology, game design and computer science. However, numerous other projects in the lab, while also arising from the research interests of lab members, were steeped in specific goals attached to larger government funding agencies.

There is no straight path to any specific research vector, rather the combination of personal interest, available funding and resources mixes to result in a posteri legitimation for action. This is not to say that researchers do not design and plan for specific projects and goals, but that the basis for those objectives to be conceptualized in the first place is the result of highly variant cross currents of technical and cultural conditions.

Once the initial idea is formed, a significant amount of preparation is required for the success of any software development endeavor. Generally, a game or piece of software will have an initial specification or design document outlining its purpose and goals. However, in academic and professional contexts alike, this initial design is generally subject to change based on experimental or prototypical results after feedback from users and other stakeholders.

At the beginning of development logistical concerns for the project are addressed. This includes initial team formation and delegation of design and technical responsibilities. Interested lab members, based on their current responsibilities, interest and time commitments will join the project. If the project is the result of a grant or other targeted research funding, specific people will already have a research and financial commitment to the project. Some projects, like *Prom Week*, may not have institutional financial backing and thus those working on the project are motivated mostly by personal interest and the potential professional prestige associated with helping advance their given discipline.

Team formation includes activities such as organization of mailing lists, initiating online document collaboration, scheduling regular project meetings and milestones, and initial design work. Technical considerations, such as the choice of development tools, programming languages, platforms, and source code management systems may also be decided at this phase. However, there are times when technical implementation details only arise after numerous iterations of prototyping. Initial development work may be devoted to creating demonstrations of various technical systems. In *Prom Week*'s case, the underlying social interaction system had been prototypically developed before the beginning of work on the game. It is not uncommon for some technical systems to be in initial or even advanced stages of development before work on a specific game begins. Due to the portability of code bases, substantial amounts of previous work or research may be reused or rewritten to accommodate new development requirements and conditions. Technical implementation strategies will be discussed further in following sections.

### *Development Strategies*

Initial design work in software development can be described in terms of two different methodological structures. The first is the waterfall method of software development. In waterfall methodologies, each development stage and its attendant responsibilities, cascades into the next like a "waterfall." The first stage of this development process is the creation of a full specification or game design document. All the features and design of the program or game are decided beforehand. Development then commences based on this initial master plan. Development milestones are scheduled in advance and usually focused on specific dates and timelines. The structure of this methodology is very rigid and predefined. Only after considerable technical and creative effort is the result played, experienced or tested. There is less room for improvisation and change. As a result, a competing methodology based on iterative design is also commonly used.

Iterative design methods are highly varied in structure and go by different names depending on the exact constitution of the iterative cycle. Regardless, iterative development

methodologies assume that time estimates will be incorrect and that the constraints of a project will change over time. As such, these methodologies express skepticism towards the orderly construction of software and therefore base development on the creation of simpler, more adaptable programs. Work is dedicated to the development of a minimum set of features and time commitment is estimated based on the perceived difficulty of any specific component of the software. This allows for specific features or structures to be scrapped or added without upsetting some larger master plan.

Academic development can follow either of these two strategies, or perhaps other less popular ones, but iterative development is more in line with previous scientific processes. Since the development of a game in an academic context is, in our case, to reify some theoretic construct, it is not always known in advance what the direct tangible result will be. In working through the initial conceptions of *Prom Week*, the development team only knew that they wanted to embed some of their existing work on interactive narrative and social simulation into a game. The genre, aesthetic or topic was not decided in advance. It was assumed that iteration on the game design would result in something revealing and engaging. The iterative design process will be the predominant focus of the following sections. Documentation specifically relevant to waterfall strategies will also be mentioned but not with the same level of elaboration.

### *Idea Formation Documentation*

The start of a project involves a considerable amount of preparatory documentation; some intended for the project and others the result of the project's intellectual lineage within the lab. Formal preparatory documentation includes: design documents and technical specifications, funding and grant materials, administrative documents, meeting notes and schedules, personal notebooks, collaborative online files, email lists, project websites and related blog posts, and project management documentation and services. Secondary documentation includes personal email correspondence, chat logs, social media interactions (like Facebook, Twitter, etc.), personal web sites and ephemeral online services.[2] The delineation here is between items that are used to document the process for the development team and those that are documentary effects of the development team's efforts. Potentially intimate, yet important documentation, like personal correspondence and social media activity, confront privacy issues, but are also an effective means of determining the daily course of research and development.

As mentioned above, the development of a research game may be motivated by previous game designs or academic theories, as such it is not uncommon that some parts of those designs and systems will be incorporated into newer projects. Another class of documents is then records of previous work incorporated into a new project such as, earlier academic publications, source code and software libraries, software programs and demos, and conference presentations. This documentation is used to initiate a new project or that has direct relationship (through project researchers) to the current intellectual task. It is limited to work produced in the lab or by team members. In *Prom Week*, the Comme il Faut (CiF) social AI system already had a demonstration implementation constructed for earlier conference presentations. Its existence provided the technical legitimation for the system's expansion into a full-fledged computer game. Some of the program code and design was thus already present in initial project documentation and cannot be ontologically separated from early *Prom Week* work.

2 Online web sites devoted to throw away tasks, like scheduling a meeting or hosting an early game demos. These will be discussed later in "Documentation Methods and Types."


Prom Week physical prototype.

## Physical Prototyping

Some game designers use physical prototyping for early design iteration and player feedback. Physical prototyping involves the construction of physical analogs for components of a game play system and emphasizes design failure and testing. This form of prototyping may not be a part of all development strategies, especially with non-entertainment software that does not have a graphical component. However, since most games and user interfaces require visual feedback[3] it is usually possible to prototype small parts of a larger system.[4] Physical prototypes are usually low quality, quickly designed demonstrations of game play systems or visual design. The intention is to make something that a designer will feel comfortable breaking and recombining in reaction to players' feedback. The immediate feedback enabled by using quick, physical mock-ups is the main advantage physical prototyping. Concerns about engagement, entertainment value, comprehensibility of game play, flow of game play experience and feasibility of design are all considered at this stage. It is not uncommon for designers to drastically change game elements based on the results of physical prototyping tests. The positive side is that the amount of work required for the prototype is kept to a minimum and is generally significantly less than the effort required to code a digital version with the same game play fidelity.

Physical construction involves converting game play systems and game design tropes into common, cheap materials, like paper, cardboard, and general crafting supplies. Some prototypes will use dice, tokens and other board game related pieces to elaborate on the

3 Some games do only use sound for their game play representations and interactions and are also amenable to physical prototyping techniques.

4 There is some debate about the utility of prototyping highly interactive game designs, like complicated first person shooters. For approaches and opinions on physical prototyping see (Fullerton, 2008), (Schell, 2008), and (Braithwaite and Schreiber, 2008).

prototype's design. Malleability of material is the core concern, as the balance, experience flow, and engagement potential of a game are rarely determined without player input and feedback. Detailed design documentation is generally less able to capture the dynamic nature of a game in play and can miss obvious procedural and emergent design problems. Iteration on the foundational systems of a game or interface is crucial in cementing its connection to a user base or audience. The process of allowing players early access to a game for feedback is known as playtesting and will continue in some capacity for the duration of a project.

### *Prom Week Physical Prototype*

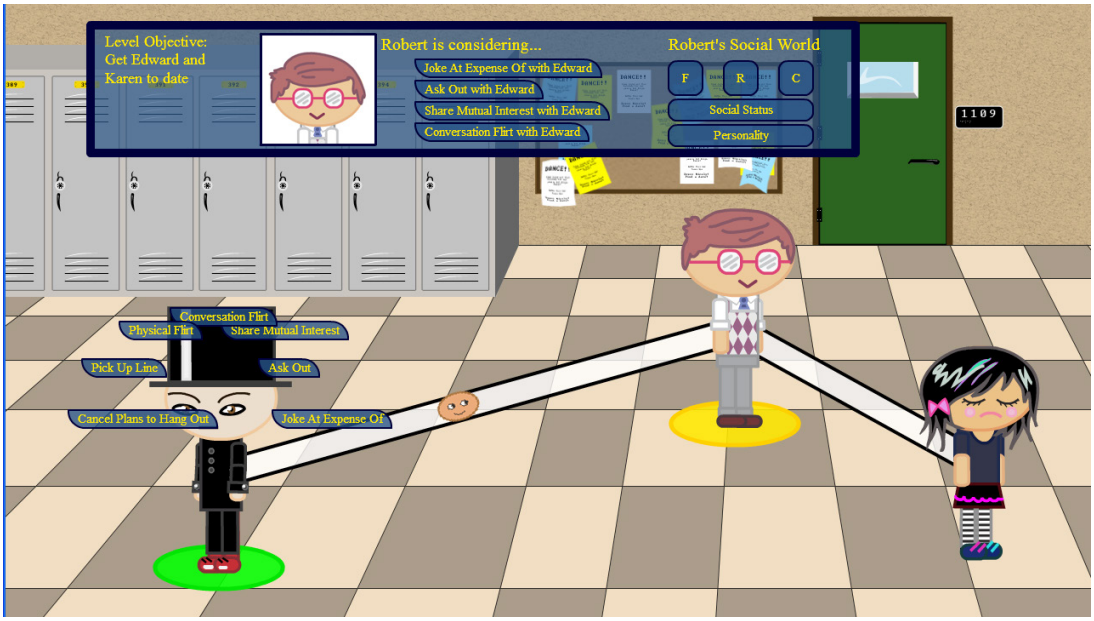
Physical prototyping on *Prom Week* began roughly a month after initial team formation. At that point it was not known what type of game *Prom Week* would become and it didn't have a name or plot. Prototyping was necessary to figure out how the game could functionally integrate with the AI systems that had already been developed. The graduate student team and its advisors met regularly to discuss potential topics and social situations ripe for the game experience. Eventually the team decided that high school social dynamics would be a good setting for modeling individual social interactions among computer controlled agents (simulated high school students). Because McCoy had already developed a computational model of the AI system, it formed the backbone for a mixed physical / digital prototype. The prototype cast the players in omniscient roles, giving them control of one of two groups of alternative high school students ("Emos" and "Goths") vying for influence over the prom's DJ, Milton. Players took turns playing cards to simulate different AI system states in an attempt to influence Milton's mental state and sway the music selection in their favor. Results of played cards were then fed into the computational system, which would calculate the results of each player move. The lab's advisors, visiting game designers and industry professionals played the prototype and critiqued its gameplay. Feedback was incorporated into newer versions and designs. The current *Prom Week* game differs significantly from its earlier demos, and its final form was influenced by the early prototyping sessions.

### *Physical Prototyping Documentation*

Physical prototyping documentation is non-standard and haphazard. Prototypes generally include quotidian materials that are hastily and heavily used through playtesting. The resulting remains of a prototype are important documents illuminating game design process, but will be difficult to ingest into permanent collections. Materials commonly include construction paper, index cards, playing cards, materials from board games and numerous other craft related materials. Documentation of the playtesting will also be contained in development and personal notebooks, online collaborative documents, playtesting evaluation forms, game play instructions and various forms of physical and electronic correspondence. While it is possible to store the remains of a particularly important physical prototype, it may also be possible to gather documentation about the design and art assets associated with the prototype as a substitute. The important thing is to document how the system played and the specifics of its rules. The low fidelity of the physical prototypes makes their corporeal remains less significant than the ideas and systems they describe. *Prom Week*'s prototype featured cards, a paper game board, and character sheets, all of which were designed in Adobe Illustrator, a vector graphics program. This allowed the prototype's assets to change on a whim with the development team printing out new ones as needed.

Early prototype work can also influence the documentary organization of the project. Prototypes are usually the first objects created for a specific game project and they necessitate the organization of planning, scheduling, and online sharing facilities for the project. This notion had a significant effect on the naming scheme of *Prom Week*'s documentation. Since

development of the initial prototype idea occupied the first two months of development time, all the group mailing lists, source code repositories and online collaborative folders share the prefix “altprom”. This is derived from the theme of the physical prototype whose teams represented Goths and Emos, two “alternative” groups of high school students. The name remained for the duration of development, being too entrenched to change with each shift of the game’s narrative focus.




Early Flash digital prototype of "The Prom" created for the Foundations of Digital Games Conference in 2010

## Digital Prototyping

Digital prototyping follows a similar process to physical prototyping except for the involvement of computers and programming knowledge. Digital prototypes are developed as playable demonstrations of a proof-of-concept, a novel technical innovation, or a core mechanic of a game.[5] Digital prototyping often follows physical prototyping, allowing developers to use insights gained in the physical prototype to create a more refined digital implementation. Committing a game design to a computational prototype requires a significant amount of effort and time, so mitigating unnecessary development work is key. That said, it also possible that both forms of prototyping happen in tandem, or that digital prototyping fails and requires the team to restart initial design work. Digital prototype development may also begin the organization of the project's technical infrastructure including source code management, digital asset pipelining, workflow management, and technical design decisions like the development platform, tools, and potential third-party resources. Or these may be specific to the prototype and not carried over into main development.

### *Development Environments*

Setting up a development environment, the software infrastructure needed to write other software, can become very complex and intricate even for engineers and researchers. Any digital game or software requires writing code and that code needs to be stored somewhere. Most modern software projects involving collaboration set up source code and file management procedures to ensure consistency and reliability across the code and asset base. One type of software routinely used to track the versions of different project files is version control software such as Git or Subversion. These software packages create repositories, either on a shared development server and/or an individual's computer, into which programmers and designers deposit source code files and any other digital assets required by the program. Version management is necessary for larger projects because there may be multiple people, in different geographical locations or time zones, working on the same group of files or code at the same time. To prevent developers from continuously overwriting

5 Core mechanics are the defining game play elements of a video game, they are the most basic and necessary actions needed. For example, a core mechanic of Tetris is controlling shapes as they fall.

each other's work, version control software requires that developers check out files from a repository. This means making a copy of a file from a repository and then editing it locally. When a developer, Carol, is done working on a specific set of changes or writing a particular piece of code, she checks the file back into the repository. The version control software then checks the incoming file against the one in the repository and, if the incoming changes are the most recent, overwrites the repository copy. However, if some other developer, Bob, had modified and committed changes to the same file before Carol saved her work, then Carol's incoming file would be based on an outdated version of the repository's code. This situation is referred to as a "conflict," due to potentially conflicting versions of the file. In this case, version control would make Carol review Bob's changes and potentially incorporate them, ensuring that the file in the repository now reflects both Carol and Bob's work.

### *Development Platforms*

Development of a digital prototype requires the choice of a development platform.[6] A platform is the collection of dependent hardware and software needed to run a specific computer game or piece of software. Platforms can be hardware-based, software-based or both. For example, in order to develop software for Microsoft Windows 8, one needs a computer that can run Windows. This computer needs to have a specific configuration of hardware, most specifically a certain type of processor, for Windows to run. Choosing the Windows platform also requires the use of software that can modify files usable by the Windows operating system. Therefore, specific developer tools and other software needed to create assets for a computer game are tied to the requirements of the Windows platform. An example of a more circumscribed platform would be the original Nintendo Entertainment System (NES). The little gray box had a very specific set of hardware and software constraints, including the use of game cartridges that players needed to physical insert into the system. This interface is specific to the platform and necessitated a specific programming language and program size to which all Nintendo games had to conform. In the examples here, Windows is more of a software-based platform, the Windows operating system is the main constraint and interface for the development of a Windows compatible game. The NES, on the other hand, is a more hardware dependent platform, as it requires all code to conform to the specific organization of hardware inside the machine. There is only one type of NES, but there are thousands of hardware configurations that can run Windows.

*Prom Week* was developed for the Adobe Flash and Air platforms. These are software-based platforms in that they are created to support a specific programming language, ActionScript 3.0, and require specifically installed software to run. Flash and Air are themselves software programs that run on multiple operating systems and inside computer Internet browsers, like Microsoft Internet Explorer or Google Chrome. Therefore, any system configuration of hardware and operating system that can run the Adobe Flash platform or an Adobe Flash compatible browser, can also run a piece of Flash software. This abstraction away from the underlying computer hardware, and even operating system, is one benefit of software-based platforms, because they can run on a large, diverse variety of computational hardware. However the ability to run on multiple types of machines also means that the solution is less optimized for any specific hardware configuration, and less optimized generally means slower execution time.

The choice of a specific development platform incurs the use of specific programming languages and asset management strategies. As mentioned, to write Flash programs, one

6 Development platform and development environment are two distinct terms. A platform is a destination for a specific piece of software, whereas an environment is a collection of support tools used to create the software for a specific platform or platforms.

has to use the Actionscript 3.0 computer programming language, no other language has the ability to interact with Adobe's Flash software and produce a game. A game's main code base also interfaces with other types of software, like databases, and requires specific file types for assets, like art and sound. These dependencies are considered when choosing a development platform, since developers and designers will be held to a platform's constraints for the duration of a project. Digital prototypes then also function as a first test of a development platform decision. Some platforms are also good for more rapid development of specific types of software. Adobe Flash began as an animation program, and therefore creating moving objects and quick graphical interfaces is a benefit of choosing the platform. It is also possible that an initial prototype will be made with a different platform than the final game. Digital prototypes are usually constructed quickly to solidify the feasibility of a design and allow quick iteration if development decisions do not work out.

Another result of digital prototype development is the organization of a project's digital asset pipeline and workflow. A digital asset pipeline is essentially the process through which a specific asset, such as a sound or drawing of an in-game object, gets inserted into the game. Many people associated with development might not be technical experts or programmers, however they still need a means to easily insert their contributions in the computational object being created. This organization is not always simple, and many projects suffer from a lack of thought out collaborative workflow procedures. Additionally, pipeline processes may not be used in to the prototype phase. Asset creation is a significant amount of work and requires consistent iteration before finalization; as a result most prototypes rely on stand-in or draft quality assets.

### *Digital Prototyping in Context*

Academic game development is often focused on the demonstration of a novel idea or proof of concept. Therefore, a functioning digital prototype of a system may actually be enough to fulfill the specific goal of the project and to support a specific hypothesis. A digital prototype of a system can also be the source of academic publication as the major outcomes may already be shareable. Academic development is a different epistemological frame from professional development due to the lack of a direct profit motive. Academic projects do not want to waste assets and resources, but sometimes failures may also be valid fodder for publications and community comment. Almost every stage of *Prom Week*'s development, including early demos, was a part or the complete focus of conference and journal publications.

Academic development does not necessarily stop at the demonstration phase, but further development is led more by the drive for verifiable and progressive results, available funding, and researcher interest. An academic game designed to showcase a new design methodology or game design trope may not need further development to make its case, however other projects might actually want to continue development due to the need for feedback from a larger community of players. Due to the need for large amounts of player feedback, crowd sourced initiatives sometimes rely on a release-able computer game to meet scientific or design-based goals.[7] The decision to commit to a fuller development cycle, and share the results with a larger, public audience is dependent on the disciplinary context and research goals.

7 Luis Von Ahn and Laura Dabbish's image labeling game (Von Ahn and Dabbish, 2004), and the University of Washington's Foldit Protein simulation game (http://fold.it/portal) are examples.

### *Digital Prototyping Documentation*

Digital prototype documentation is mostly technical, and functions in tandem with previously mentioned document creation sources. Digital prototypes are smaller, less fulfilled versions of some larger, more complex technical goal, but they still often involve the need for organization of software development tools and workflows. It is at this point that the inherent complexity of technical development documentation comes to the fore. In documenting a digital prototype, an archivist may come across records for development environments, prototype iterations, platform specific resources, third-party software and libraries, custom peripherals and hardware, and version control and development management tools. Because of the digital prototype's existence as a software object, much of its documentation will be born-digital content based on either compiled binaries or source code files.

The digital prototype itself will consist of creative assets, source code, executable binaries, third party libraries and potentially, custom hardware and peripherals. If the project used version control software, then most of the important files will be stored inside a code repository on one or more machines. These repositories then are the main source for technical project documentation. However, due to the chaotic nature of prototype development many important early files may either precede version control's implementation or be viewed by the development team as unnecessary for inclusion in later documentation. Additionally, many game assets or drafts of assets will not be included in source control because they change infrequently or are only used conceptually or for early promotion. *Prom Week*'s online collaborative folders contain thousands of game related and technical documentation files not used in the final version of the game. Early work might be scattered across many locations because it occurred in the period before a stable research goal and development trajectory formed.

Digital prototypes may also be used in preliminary research or in academic knowledge creation. In these situations, journal publications and drafts, user studies, conference papers and playtest evaluations will also be produced and be crucial to the interpretation and success of a prototypical work. *Prom Week* was a fixture at game related conferences, like AIIDE and FDG throughout the course of its development.[8] Major work on *Prom Week* was tied directly to conference and publication deadlines, at times requiring significant alterations to the underlying artificial intelligence and game play models. In *Prom Week*'s case, the basic development infrastructure of the project, and its Adobe Flash platform, remained consistent throughout the project. Milestones and publications may, however, cause changes in platform or direction drastic enough to significantly affect the consistency of the documentary record. Failed demonstration work is less likely to be saved if it is not part of the retrospective, teleological document trail leading to a successful publication or presentation.

8 Artificial Intelligence and Interactive Digital Entertainment (http://www.aiide.org), and The Foundations of Digital Games (http://www.foundationsofdigitalgames.org) conferences.

## Development Methods

Development processes differ significantly based on initial project goals and the choice of development methodology. As mentioned above, development strategies exist on a spectrum with rigid, totalized planning on one end, and loose, iterative cycles on the other. Every possible permutation of development process will not and cannot be described here. The focus will be on the general contours of game development in academic contexts, as an example of innovative cultural software, with a nod towards the developmental and documentary implications of specific caricatures of development processes. Development constraints are also defined by a project's specific goals, especially if they are based on previous, established work or funding agency requirements. Different development practices support different types of academic structures. For instance a game designed as an implementation of a clinical process for use in a medical trial will have a more rigid development structure than an experimental game designed to highlight a new game play innovation. In general, iterative design practices and playtests will almost always occur since the goal of game development is to produce a playable object. The influence and necessity of testing is dependent on the nature of the project at hand. Discussed below is the general structure of academic game development. It follows a more iterative developmental approach in line with the analysis of *Prom Week*'s development.

The previous prototyping phases may not be included in a development process, especially a totalized process like waterfall development. Regardless, the beginning of development will usually involve the creation of a game design document or some basic hint of a specification. In waterfall development this document dictates all facets of the game design and is very specific about the needs and goals of game system. In iterative development, as the game has usually developed from prototypical phases, there is usually less of a focus on a single, unified document dictating every possible and proposed feature. Instead, features and game play designs to be implemented are ordered in priority and assigned to team members. These assignments are then the focus of concerted development effort for a short interval, usually no more than a couple weeks. During each iteration interval features are implemented, or problems in implementation arise and are analyzed and changed. Iterative development relies on the ability to change features and direction quickly as problems and pressures arise. Specific iteration intervals also make sure that certain problems always receive attention quickly, instead of straying off course due to lack of oversight or testing.

*Prom Week*'s development followed an iterative strategy, but the resulting workflow and progress were not as clean cut and flexible. Academic development is conducted by graduate students with varying levels of professional development experience and undergraduates generally possessing none. Therefore the implementation of a clean, fixed development process is constantly in a state of flux. *Prom Week*'s development leads managed their undergraduate programmers with the implementation of weekly Scrum meetings.[9] Each team member was responsible for a specific development task or feature and worked on it exclusively until the next Scrum meeting, after which progress was analyzed and goals reoriented. In some cases developers and undergraduates engaged in pair programming (in which two people work together on a single screen) or in-group programming sessions, where many people would collaborate together in the same location. The main development leads, however, did not follow such a tight management methodology. The game play design of *Prom Week* remained quite fluid for the first year of development, and each graduate

9 Based on the Scrum agile development methodology. Essentially a way to structure project work into short "sprints" dedicated to different features. The "sprint" is a single unit of time, usually a week or two, after which the work and goals are re-evaluted. This prevents spending too much time on difficult or potentially wasteful tasks, in theory.

lead took ownership of specific aspects of development, but deadlines and time availability caused distinct crunches for conference and publication demonstrations and a diffusion of specific development goals over the entire team.

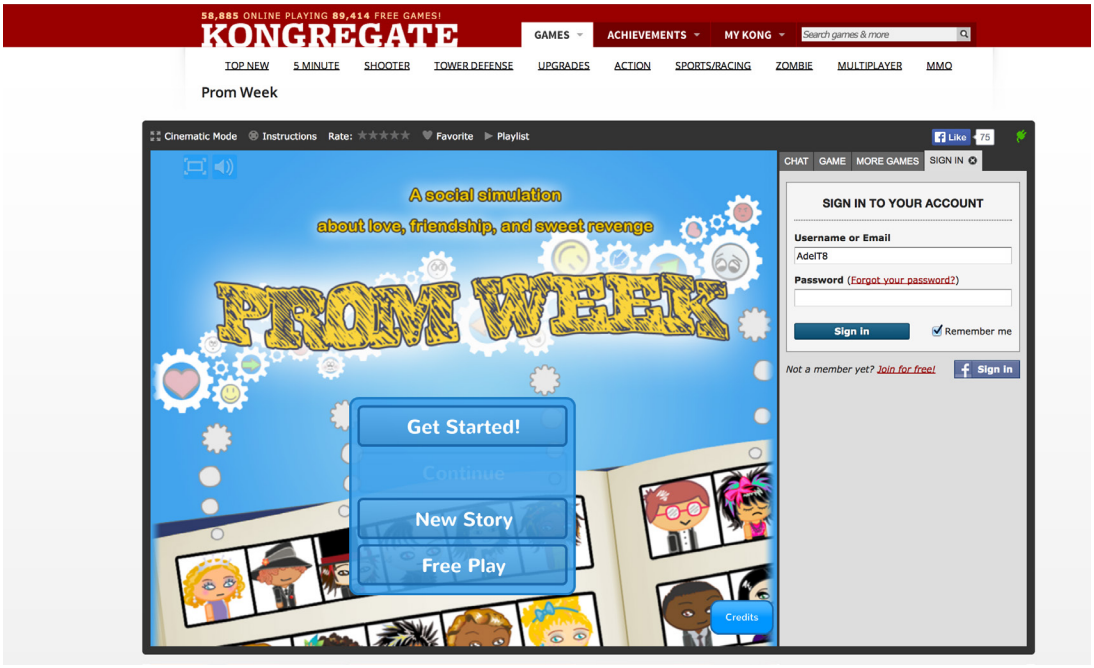
In general, development milestones are tied to either the completion of specific sets of features or to time sensitive demonstrations to larger communities of practice. Academic development is heavily tied to the cycle of scientific publication, conference presentation, and progress reports to granting agencies. A project's purpose, validation and legitimation are derived from the dissemination of and commentary towards the work's technical and theoretical accomplishments. Specific game development milestones are based on the level of completion of the game's played experience. Alpha versions or builds[10] are considered to be nearly feature complete, in that most major technical development is finished. Beta versions are content complete, in this state most of the major art and sound assets are finished but significant bugs, low level design modifications, and balance issues remain. Content creation sometimes relies on the completion of game features and is usually finalized after the completion of the game's technical systems. Pre-release versions of games are then tested extensively to locate bugs and fix final issues before release.

### *Development Methods Documentation*

General development process documentation is similar in kind to that produced for a digital prototype. Most of the documentation organization needed for idea formation, and physical and digital prototyping, is extended for use during the full development cycle. Versioning of source code and asset files is handled in a similar way and online collaboration is managed with the same infrastructure, however the operational scale is usually much larger. The volume of documentation will increase through dedicated work on a project. Since publications, presentations, and testing continue throughout development, academic document production will mirror development advances. Increased production towards a final release also involves more people and collaboration, with art, sound, and content creation work proceeding at a level much greater than was needed for prototype construction. A routinized development process may also be put in place, so design documents, meeting notes, and progress reports will become more common.

On the technical side, full development processes incur additional overhead needed to integrate even more code, systems and features into the final product. Testing procedures, needed to make sure that a game's code is bug-free, require the use of testing frameworks, and systems for reporting bugs to the development team. Common development support software includes version control, bug and task management, and unit test frameworks. While the support software is not necessary to run the final product, it will contain a significant amount of information about specific development and implementation decisions and related issues. Additionally, the development of a complex computer game requires the simultaneous development of programs to help with content generation and the digital asset pipelines. These custom development tools will require the same types of archival appraisal and scrutiny as the primary game object.

10 A build refers to the result of compiling source code and assets into a playable piece of software.




Prom Week title screen as hosted on Kongregate, a web browser based games portal.

## Release and Dissemination

A software or computer game release represents the demarcation between the internal development process and the external experience of the game by a more general audience. Up until release, most of the documentation and design decisions hinge on the development team and the small community of those who have playtested and critiqued pre-release versions of the software. After release, the development shifts to accommodate feedback from a larger audience. More specialized game software may only have a limited release to a specific community of practice with more nuanced and targeted feedback. General release games exist in larger global media context and may receive international press attention and criticism. That said, due to the lower budgets and more limited focus of most academically produced software, it is rare that an academic title will garner an audience as large as a professionally produced title. Regardless, the types of attention received by even a modest academic release will mirror larger, professional productions in the modalities of new documentation and its influences on the development process.

Release typically occurs after the game has reached a state where it is content complete (though content scope may be revised and reduced in order to make timely release possible) and relatively free of bugs. The drive to release can create immense pressure for a development team, especially if the release date is not under the team's control. In academic production, many projects are tied to fixed funding intervals and short-term grants, so the final product might be released prematurely or not at all. Preparation for release includes deciding on dissemination methods, public relations, and continued maintenance.

### *Dissemination*

Dissemination for a game release can take on multiple styles and approaches. Professional titles are often physically sold, which involves production processes and incurs a large amount of capital investment, or downloadable from online stores. Academic games are generally made freely available online and often hosted on university servers. An academic release, then, is simply making a game available a larger user base; it will not have a large marketing effort or be available in stores. From the perspective of the developers, release is an important moment when their work will be tested and analyzed by new players. Numerous commercial websites will host games for free in exchange for the traffic generated

by new players. Dissemination can then occur through either university-supported channels, or through upload to third-party websites. *Prom Week* was released to the public on Facebook and Kongregate, a large game sharing community site. Facebook provides a framing for games hosted elsewhere (in the case of *Prom Week*, on university servers). Kongregate hosts Flash and HTML / JavaScript based games from independent and small-scale developers. Release in dual contexts required the *Prom Week* team to push out post-release updates in two different ways.

Software dissemination requires the organization of a release package for the target user and system. In the case of web-site distribution for *Prom Week*, the only requirement is the Flash game itself (a SWF file incorporating all compiled code and game assets). Software developed for a specific platform or operating system, like Windows or Mac OS, needs to be downloaded and installed in order to run. This installation process requires the use of additional software to put all necessary components into the correct place on the host system. Typically installation is handled by a single installation management program set up by the developers to ease the installation process. However, many academic releases may require more detailed and arduous installation methods if there was not time or funds available to simplify them.

Academic game releases can also have a significant effect on the game's research component, if present. Some releases will be orchestrated with the singular intent of collecting player-produced data for later study and analysis. These streams of data can incorporate play traces (records of the actions a player took in a game), simple usage statistics, or data generated by the game itself as a solution to a research problem. The data generated by a software product after release can be important to its position in academic research, and form the foundation of further publications or studies. Scientific inquiry will usually continue after an academic research game's release, meaning that the release itself is a catalyst for more research. For *Prom Week*, the developers created a server-side component to monitor game play data and statistics, this data was then used in later publications and even for non-*Prom Week* game play analysis research.

### *Public Relations*

A public release entails interaction with the larger gaming community and press. Public relations strategies, including social media management, promotional web sites, and press releases, may be employed to generate excitement and players for a game. The audiences for academically produced games are generally small, though some games produced by more vocational game design programs (and not intended for research) can reach larger audiences. *Prom Week* is an exception in that it was nominated for independent game design awards while simultaneously the object of ongoing artificial intelligence and systems research. Interactions with the press may include interviews, game reviews and previews, awards, and industry conference presentations. The game's associated institution may also become involved in the promotion of the game and mobilize its own internal public relations resources.

### *Continued Research*

If there are no future problems or continued research interests, then release represents the end of development for most of those involved. This is also a time of great risk to the preservation of development records, as team members may already be moving on to other projects and wholly abandoning maintenance of the code repository and asset management services associated with the project.


Prom Week promotional one sheet

### *Release and Dissemination Documentation*

Public relations documentation is the major documentation output following a release. This documentation includes press releases, social media commentary, promotional websites, and online articles and reviews. Many people may have played an earlier version of the software, but the release build is supposedly the most stable and best representation of the design goals of a game. Online and social media reaction will be contained on game related and personal blogs, professional technology and game websites, and social media services like Facebook and Twitter. The university or associated institution will probably issue press releases and will usually host the released game. Most academically produced games are available for free, as they are the result of government or institution grants and are not created primarily for profit. Additionally, there may be final administrative documentation such as final reports, academic publications, licenses, and patents necessitated by the general availability of the game. Further documentation, in the form of fan sites, online frequently asked questions (FAQs), gameplay videos, game walkthroughs and guides, and machinima works, may also arise if the game reaches a significant level of popularity. Released games, especially those with novel technical innovations, may also receive scholarly and informed treatment from other academics and developers. Games representing advances in specific fields will be cited in journals and monographs by individuals not associated with development.

On the technical documentation front, installation procedures and software packages, digital distribution web sites and services, and distributed release binaries, are the main items associated with release. The technical and project documentation is also now at greater risk of abandonment and obsolescence due to a shift in academic and professional focus

among the development team. Academically produced works, if they received federal or government funds, will probably be subject to enforcement of a data management plan requiring all research output and documentation to be stored in some permanent location. Those projects made without direct oversight will probably lack any coherent preservation or post-development organizational strategy. It is likely that source code repositories and online collaborative documentation will remain available in the immediate term due to the mirroring of data across multiple computers, and the trivial or free cost of small scale document hosting.

## Refactoring and Continued Development

Although the release of an academic game may signal the final effort of the design team, it is more likely that significant aspects of the game's development or its associated research will be conducted after release. Post-release modifications are common considering the complex nature of software products, and there will probably be a significant number of subtle bugs in the system that may need additional work. Other post-release work might be needed if the game in question is accepted into further conferences or awards consideration. Because further accolades and publication draw attention to the work, it is in the developers' interests to continue to improve a game to ensure that new players get an optimal experience. Modifications and their requirements depend on the type of issue being addressed and the platform and method of software distribution for the work. Post-release work either consists of bug fixes (modifications to acknowledged problems with the software) or refactoring and redesign. Refactoring involves rewriting and changing parts of the interface and code base to improve performance, graphics, or underlying code coherency. These changes may not be noticeable for the average player, but may constitute significant improvements for anyone responsible with long-term maintenance.

After a technical or cosmetic change to a game, a new version must be distributed to users so that they can remain up to date and avoid conflicts with older instances. If a program is installed locally, it is then dependent on some procedure for downloading and replacing outdated components. Generally referred to as patching, smaller updates (patches) are downloaded either by the player or the program itself. New updates overwrite and erase previous versions unless specific facilities are provided for their retention. Patches also update the version numbering of the software, providing discrete tracking of the state of the installed program. Games distributed via web browsers may also have versioning information, but do not allow the user to control when and if an update should be installed. When a game is embedded in a web page, the browser will download a fresh copy of the game's data every time it detects a change. This process is usually seamless and unavoidable. As such it is difficult to record updates to browser-based games, since there may be no explicit record that a change has occurred. Typically the developers will upload or submit an entirely new copy of the game to an online host, who then replaces the old version and immediately begins serving the new. In this case, having access and control of the development's source control repositories is paramount to maintaining a record of software modification.

Another reason for continued development is that the game is a vector for an ongoing scientific study. The release may be to a group of users about whom the researchers have a specific interest or research goal. Data generated as a result of continued research is then incorporated into further work by the development team in promoting findings through publication and other means of academic knowledge dissemination. It is not uncommon for work from one project to immediately influence the development of a new one. Perhaps some aspect of the current game or its data is implying a new use or hypothesis. The academic development process is then an intellectual circuit whereby new findings improve software solutions that promote new research. *Prom Week*'s underlying social AI system, Comme il Faut, is now the basis for a much larger project focused on conflict de-escalation by military personnel in unfamiliar cultures. Additionally new versions of *Prom Week* are also being developed for mobile devices, and its gameplay data incorporated into continued studies by new researchers not associated with the game's development.

### *Refactoring and Continued Development Documentation*

Continued development documentation is similar to that of the primary development, but usually less voluminous. Sometimes a release will signal the beginning of continuous updates to a game and development might simply continue. Other focuses might be porting the game to a new platform in hopes of reaching new audiences or research goals. These new versions and modifications must also be documented in a similar manner to the original. The development infrastructure for further updates, patches and changes is most likely the same as that used for main development. A new version might change platforms or have a goal so divergent from the early work's intent that a separate version control and document management apparatus will be created. Continued development will incur patches and software updates, user account and user produced data, ports, and further academic publication and related documents. As an example, an iPad version of *Prom Week* was attempted and started an entirely new development code base.

Games generating data streams will also need special consideration, since they may remain active for years after an initial release. Perpetual data streams from games are a relatively new phenomenon and their storage is a significant issue for future archives and libraries. While the format of data in streams is similar to other scientific outputs highlighted by JCAST and other reports, they only have recommendations for data collected in intervals or at the cession of research. Continuous data streams are not mentioned. Streamed data can also depend on proprietary analytic software developed by a research team. Care should be taken in saving data analytic tools associated with a research game or program. Data streams are also associated with online distribution platforms, as certain websites give developers access to user statistics and analytics for hosted game content. The documentation can include proprietary experimental data, game play traces, and first party and third party usage statistics. Experimental and study related information may also be located in a separate location from the main development documents, as a study with a software object is usually not dependent on or related to the technical development documents.

# Project Organization in Context

The Prom Week primary development team (from left): Michael Treanor, Aaron Reed, Joshua McCoy, Benjamin Samuel, Michael Mateas and Noah Wardrip-Fruin.

## OVERVIEW

All cultural software is created in a specific social and organizational context. As we found with *Prom Week*, the work of academic game development and its documentary production do not occur in a vacuum. The norms of the academic community, the development team, and individual researchers influence research and development progress. This section provides a social and organizational context for the motivations, constraints, and commitments of games researchers and a sample of the lived reality of academic game development and scientific study. The organization of a graduate level computer science research lab, how personnel relate and interact, and how the members of the development team communicate with external peers and industry are outlined below. Knowing how individual researchers relate to the development process, their institution and each other is important for understanding the contents of a development archive. Understanding social structures can also direct archivists and librarians to the correct people when questions or confusions about cultural software documentation arise. Much of the information here is generalized from interviews with the *Prom Week* development team, but the general structure is likely similar to other departments and labs.

The following sections elaborate on the general institutional relationships and organization of project team members, lab directors, administration and external communities. "Laboratory Organization" discusses the position and concerns of an academic research laboratory in the computer sciences. "Personnel and Structure" explores the structure of interpersonal and professional relationships of lab members. The "External Communities" section focuses on institutional and lab relations with academic conferences and organizations, and they function as targets of knowledge dissemination and publication. This section also looks at relationships with scholars and practitioners outside the lab and others in industry. The social organization work here is followed by a deep documentary analysis of the development process now framed through the institutional structures provided below.

## Laboratory Organization

The laboratory is a structure inside the modern research university devoted to a specialized scientific or technical tradition. It does not have to focus exclusively on one area of study and can be quite multidisciplinary, however the core values and intellectual valences of its inquiries are always somehow related. Individual research labs exist in a hierarchy and are affiliated above with academic departments and schools and below with specific researchers and students. Administrative decisions and the interests of lab members merge to dictate the course of research. The lab may also be part of a larger, non-departmental organization of multiple different departments and labs across a campus or multiple institutions. The Expressive Intelligence Studio (EIS), the lab responsible for the development of *Prom Week*, is a part of the Center for Games and Playable Media at UC Santa Cruz and also affiliated with the Computer Science Department in UCSC's Jack Baskin School of Engineering. Both of the lab's co-directors are Computer Science professors and the lab's initial focus was on expressive artificial intelligence systems and interactive narrative techniques. Over time the lab evolved to include a digital humanities component, focused on computer game studies in addition to maintaining a research agenda in the field of interactive artificial intelligence.

An academic lab's primary purpose is knowledge generation. Journal publications, conference presentations, posters and workshops, and novel technical systems or discoveries are the paramount outputs. Publications bolster both the reputation of the lab and its individual members, which is instrumental in fostering the development of new research trajectories. Academic prestige also attracts new members to the lab, who in turn help elaborate and foster research goals. The field of Computer Science is diverse and continually expanding. According to the Association for Computing Machinery, computing "is a broad field that connects to and draws from many disciplines, including mathematics, electrical engineering, psychology, statistics, fine arts, linguistics, and physical and life sciences."[1] Research into computer games incorporates elements of creative art, human computer interaction, and potentially deep system engineering, graphics, and simulation skills. Game design research in games promotes theories about system and user interaction and attempts to test hypothesis through system construction. Time and effort are required for scientific research and success is tied their effective management.

Funding for scientific study in academic labs comes primarily through research grants or from institutional and external sources. Most major projects undertaken are based on work proposed to various governmental and institutional funds or agencies. Grants are awarded based on records of previous work, relevance to the field, and research potential. The funds awarded support the work of graduate researchers, who conduct most research under the guidance of lab support staff and faculty. Undergraduate students also contribute, and can be remunerated with academic credit and/or scholarships, hourly wages, and so on. Funding from granting agencies is often time limited and focused on results. Research projects generally account for only a percentage of the work in a lab with others supplementing their research with teaching or administrative duties. As an example, graduate students primarily worked on *Prom Week* in addition to other research and teaching commitments. A majority of the work occurred outside the normal funding stream. Some funding for *Prom Week* came from NSF CAREER and GRFP grants, but there was no explicit funding for the project from any other source.[2]

1 The Joint Task Force on Computing Curricula, "Computer Science Curricula 2013." Association for Computing Machinery and IEEE Computing Society. This report also mentions 18 different knowledge areas within the purview of modern Computer Science practice, with Graphics and Visualization, Human-Computer Interaction, Intelligent Systems and Software Engineering being particularly relevant to games like Prom Week.

2 NSF CAREER grants are awarded to beginning faculty to help bolster their initial research program and are not directed to any specific research (this allows for initial flexibility). NSF Graduate Research Fellowship Program (GRFP) grants are

## Personnel and Structure

A laboratory's document production is embedded in the hierarchical relationships of its members. Generally, most labs at academic research institutions (as opposed to national or industrial research and development labs) are run by tenure-track faculty and staffed primarily with graduate and undergraduate researchers. Faculty members outside the department or lab can also affiliate with it if they have intersecting interests or advise lab graduate researchers. Others, such as post-doctoral staff and visiting scholars may help on specific projects or have short appointments at the university. The interactions and norms facilitated by lab members and promoted by lab director(s) dictate the research direction and potential funding sources of the lab. Each group is discussed below to provide a picture of lab relations and their potential impact on the documentary record.

### *Directors and Faculty*

At the top of the hierarchy is the lab director whose research interests and expertise dictate the course of research. A director is usually a tenured or tenure-track faculty member with appropriate terminal degrees in the department(s) affiliated with the lab. Generally a lab has one primary director, though some labs may have more.[3] In addition to founding or running a lab based on her research interests, a director also advises graduate and undergraduate students. An advisor admits graduate students into the community of the lab based on perceived research fit and scholarly and technical abilities. As a result of the collaborative research interest, most publications or lab outputs are attributed, in part, to the advisor's guidance and most published work by graduate students is also co-written or co-produced by an advisor. A director also functions as the principal investigator for any grant or research project under study or development at the lab. Since there are usually multiple such projects in progress at one time, the director's goal is to ensure that all are progressing according to grant dictated and/or internal goals. A director also assigns specific graduate students to projects based on the student's research interests and with an eye towards the student's future academic career.

Given that the director is overseeing multiple projects and orchestrating the position of the lab within the departmental and administrative hierarchy, they spend much of their time in organizational and logistical pursuits. Being faculty, they do also conduct their own research, but must also teach and mentor those lower in the hierarchy. Teaching classes, grading, and appointments to academic committees all function to reduce time devoted directly to research. In their advisory capacity, they use interactions with graduate students as a means for implementing and exploring their research agenda without the extensive time commitments required of detailed, advanced research. This is not to say that directors do not produce significant work on their own, but that – especially in labs with a significant number of projects – many of the projects in the lab will receive more attention from students (and perhaps staff) due to faculty time limitations and the desire to train students to become research project leaders themselves. In a Computer Science lab, due to the processes of development and the inordinate amount of time needed to implement and debug computational systems, it is common that the advisor will provide an intellectual approach, framing, and legitimation for the work under way instead of helping with system implementation. Advisors are most active during project and idea formation, as the basic lines of research are usually extensions of their own past or related work. Project progress is addressed through project meetings and other regularly scheduled lab events. Advisors provide feedback, critique and potential solutions to development problems based on their own expertise and experience. In many cases they will not, however, be the best source

awarded to promising new graduate students for their general research.

3 The EIS lab responsible for Prom Week has two co-directors, Noah Wardrip-Fruin and Michael Mateas.

for the documentation of the project, given that they will have spent limited time with the development documentation and code base. Directors also function to promote the lab and its work to external communities, and will use their peer and professional networks to increase awareness and dissemination of lab work.

### *Graduate Students*

Graduate students are individuals pursuing an advanced or terminal degree, usually an M.S. or Ph.D. in their chosen field of study.[4] These students apply to university programs either to work with a specific lab group and professor, or to hopefully find a specific research interest while advancing their knowledge of an academic field. Most advanced Computer Science research involves the development of technical systems or the elaboration of computational theories and constructs. Specific development or research goals are generally not set by the graduate students by themselves, but are worked out in collaboration with advisors and the available funded projects in the lab. Because of their lack of experience and lower standing in the academic hierarchy, graduate students spend much of their time in advisor directed research roles. Graduate students do not usually have official titles or administrative responsibilities beyond their membership in a program or lab, and are often explicitly paid or funded for full or part-time research. Graduate students may work with advisors to dictate lab research goals, and students with particularly strong expertise in a given area (many students come from professional backgrounds) will collaborate on grant work or research project formation. Independent graduate research can also lead to larger projects. In *Prom Week*'s case a student-developed AI system spurred the creation of a full game.

Development documentation and process may be organized in concert with faculty advisors, but the major effort of development (or granular research activities) is generally graduate student directed and organized. Due to time spent on a project, graduate researchers are probably the most knowledgeable members of a development team and will have access to the documentary resources of a project. On *Prom Week*, graduate students oversaw all the version control and online collaborative documentation resources, and wrote a majority of the project code. Graduate students also work with advisors to formulate results and findings throughout development for dissemination as academic publications. Advisors are listed as collaborating authors on most graduate student output since they participate in writing and editing most lab publications and have a hand in most projects (at the very least in project formation).

Graduate students may provide the most work on a project, but like their advisors some also have extensive involvement in other tasks and responsibilities. In order to maintain funding, graduate students may work on projects designed by their advisor or others outside of their immediate research interest. It is not unusual for a graduate student to spend half of their time on research unrelated to their eventual dissertation topic. However, quite a few also receive full time funding for specific, personal research. Graduate students are also active students, in that they take classes to fulfill degree requirements and must pass the processes required for Ph.D. acquisition. They may also teach or assistant teach as a source of income, sometimes with their advisors and in a related field, and sometimes in wholly unrelated disciplines. The members of the *Prom Week* team assisted in courses in artificial intelligence, interactive narrative, basic computer science and even humanities in addition to their research. Also, the development processes outlined in the previous section are conducted in an atmosphere of competing interests and resources. Many graduate students will rotate through projects over the course of their study and the documentary record may reflect some of this discord. Additionally, due to varying levels of involvement in a project,

4 Due to the EIS lab's focus on interdisciplinary work, some of the graduate researchers for Prom Week were pursuing M.F.A.s in the Digital Arts and New Media Program (DANM).

some team members may have no knowledge of crucial historical or technical details from development. Obtaining information from multiple project leads and team members will probably be necessary to construct a coherent timeline and provenance for available documentation.

### *Undergraduates*

Undergraduate researchers become project members for a variety of reasons: at the request of a professor, for extra research credit, from personal interest, or in some cases through funded undergraduate research positions. Undergraduates are involved with projects on a number of levels, some look for a little research experience while others spearhead important aspects of a project. Some will decide to continue towards advanced degrees based on research experience.[5] Undergraduate research publication is also possible, usually with the help of an advisor or mentoring graduate student. Contributions to projects are usually of a limited capacity in quarter or semester long engagements with very specific goals. In academic game development, due to the high degree of specialization required to architect and design novel systems, undergraduates may be assigned supporting tasks, like data entry and content generation. They may also help create automation or augmentation tools to help speed up development tasks. Since undergraduates are generally limited in their time on a project, their tasks are specific and simple enough to allow for completion in a relatively short period. Some undergraduates do actively contribute to a project for longer intervals, especially if they have significant technical or design skills, but the majority are using their role on the project as an autodidactic measure or to gain skills valuable for future employment. Most undergraduates will leave a project, graduate and get a job based in part on their experiences with project development in the university.

On *Prom Week* the undergraduate team contributed considerable amounts of the content including art assets, sound design and written dialogue.[6] In line with the discussion above, the undergraduate researchers did not implement *Prom Week*'s major research contributions. They did, however, design tools the development team used to insert social scenario content and dialogue into the game. Undergraduate involvement was sporadic, with many moving onto other engagements and graduating before the completion of the project. Undergraduates are also heavily involved in coursework and the general requirements of obtaining a four-year degree, so their time investments are much less than graduate researchers.

5 Undergraduates also lead their own research projects in research universities, though often outside faculty labs. And in other contexts, such as four-year colleges, undergraduates can play a role more like that described for graduate students above.

6 An undergraduate led the sound design for Prom Week. Even junior researchers contributions are significant appraisal targets and may be in need of preservation.

## External Communities

Lab activities are also focus on relationships with communities, companies, and people outside of the immediate lab group. In many cases, the influence of outside forces is instrumental in directing lab goals, policies and funding.

### *Academic Conferences*

Academic conferences are a major means for disseminating information about a project. In fields partially reliant on system development, like computer science and game design, conferences provide a venue for demonstration and feedback. As a result, major development milestones are tied to conference appearances. Larger demonstrations to academic peers might cause stress on the development team and lead to extra work or "crunch time." Successive conference presentations are important to generate continued interest in the work. Systems should keep improving and demonstrate noticeable progress, otherwise funders may take notice and lab prestige may suffer. In peer-reviewed funding situations, appealing to other academics can be as important as impressing the funding source staff.

Conferences vary in size and focus. Larger ones explore general trends in a field and smaller ones focus on sub disciplines. Lab members will usually participate in the same conferences throughout the years, and build up a small set of key appearances to highlight their work. The EIS lab at UCSC is involved in computer science and game design, with specialties in interactive narrative and artificial intelligence. The *Prom Week* team regularly presented work at game artificial intelligence, game design and game studies conferences. Since conferences follow a known schedule, documentation and development effort will cluster around presentations and demonstrations.

### *Outside Scholars and Practitioners*

Correspondence with outside academics involved in similar research is vital to the progress of any field. Members of a specific sub-discipline will share mailing lists, meet at conferences and extensively discuss findings hoping for feedback, criticism and encouragement. They also function in the peer review system for grant funding and academic publication and can have significant influence on a project's success. Scholars frequently visit other institutions to temporarily teach or lecture. Such visits allow for members of particular lab to receive feedback and demonstrate recent work. The community of scholars creates consensus around certain topics, argues over others, and helps further the research direction of a field. Usually graduate students and scholars at multiple institutions collaborate on large-scale projects, pooling resources and expertise.

The *Prom Week* team demonstrated prototype versions to visiting researchers and game designers on numerous occasions. Feedback from these demonstrations helped guide the course of development and validate the project to important individuals outside the lab. Many members of the game design community were given early access to the online prototype of the game before release. Making the game design community aware of *Prom Week*'s efforts and innovations was instrumental in helping motivate the project's completion and recognition by industry and press. Publicity benefitted the lab by raising its prominence as a location for game systems research.

### *Relationship to Industry*

Science and engineering labs sometimes collaborate with companies in related industries. Some companies will pay for sponsored research, especially if the lab's specialty is in a commercially viable field in which the company lacks expertise. Industry sponsorship can range from funding entire academic laboratories to targeted, short-term engagements with

a specific technology. As stable funding is an issue at many research institutions, additional streams of research dollars can help a lab cover its finances and provide opportunities for lab members to intern at or join companies that have benefited from their work. Graduate researchers sometimes interface with corporate research and development through sponsored projects as a means of future job security. Sponsored projects allow for a company to use academic resources in exchange for potential publications and increased lab exposure, especially if lab effort is reflected in later products or services advertised by the company.

Sponsored work is bound up with numerous restrictions on dissemination and access. This leads to situations where some lab members might need to keep their work from others in the lab due to non-disclosure agreements or other legal protections. Work conducted in this way is usually cleared for publication after it is made public by the industry sponsor. In computer science and game design, many successful products and services have started as academic research, leading to extensive collaboration between the computer industry and computer scientists.

# Document Methods and Types

## OVERVIEW

This section is devoted to a more detailed discussion of software and game development documentation. Games, in particular, create a diverse set of materials due to their interdisciplinary nature. Programming and art combine in games, creating a unique set of digital and non-digital assets to consider for preservation. Below we outline the different types of documentation produced in the development process, discuss potential methods for their analysis and appraisal, and note the difficulties we encountered in organizing the records. After a look at the documentation, we cover its collection into physical and digital repositories for storage and archival treatment. Many of the processes and documents covered here will be applicable to a large swath of computational objects, however our examples are based on the documentary analysis of *Prom Week* and are probably not completely exhaustive (nor attempting to be). Many funders now require detailed data management plans covering all documentary production. This section is also a guide on what to consider when organizing a data management plan for a game-based project. Recommendations are placed throughout each section, and larger, general recommendations follow the description of preservation issues and practice.

## Physical Content

The primary focus of this report is on cultural software objects, and in our investigations we focused on academically developed games. Games, in our context, are wholly digital objects, and much of their documentation and process information is born-digital as well. However *Prom Week*'s documentation did contain some physical artifacts produced in the game's development specifically. Physical prototypes and their remains are covered, but general physical documentation (like notebooks, reports, etc.) is not. We feel that dealing with physical documentation is thoroughly discussed elsewhere, and much of the documentation usually found in physical sources is now stored in digital form as well. Further, for projects that produce physical documentation, our recommendations, classifications, and descriptions in the born-digital section below still apply. For more information regarding the storage and appraisal of physical game development documentation, we recommend looking to sources in the bibliography dealing with the appraisal and retention of science and technology records.

### *Physical Prototypes*

As discussed in "The Process of Academic Game Development" above, physical prototypes represent initial designs for game play systems or user interface elements. Most instances of cultural software objects will have some user-facing components and are thus likely to have some form of prototype. For games, physical prototypes are often the first step towards creating an engaging game play system and are crucial for communicating a designer's ideas to others on the team or in the community. Therefore, if possible, preserving physical prototypes is important to understanding the course of a game's development and its history of interaction with users.

If preserving the prototypes themselves proves too onerous, try to record the physicality and interaction of the system in some other form. Even photographing the individual system components and keeping a copy of the rule set might be enough to make it possible to recreate the experience (if a future researcher so desired). Prototypes are often about the systemic interactions and what they reveal to the designers. Feedback from prototypes is probably more important to the process than the prototypes themselves. Archivists should note this distinction when appraising physical prototype records.

*Prom Week*'s physical prototype consisted of custom-printed cards, a game board, player tokens, and a computational assistant. Since the prototype is essentially sheets of paper,

we found it easy to store in a conventional folder and archival box. Due to the prototype's mixed nature, the computational component underwent storage with the rest of the born digital content. The digital design files for the prototype were also stored in case the physical remains become unavailable.


Prom Week local press coverage and promotional t-shirt.

### *Physical Ephemera*

Some other physical artifacts include promotional and press materials created for conferences and public events. Teams may have t-shirts, pins, and other forms of conference memorabilia. Appraisal for these items is difficult because they are sometimes awkward in size or shape, or contain non-archive friendly components. An attempt should be made to record their existence in the development history, but in most cases a photograph of a t-shirt and shirt itself will have similar value for historians. This is a rather under-researched area for archival work and we will leave the discussion here with a recommendation of further study.

### *Digital Records of Physical Ephemera*

Many of the prototype materials and press promotional materials may be designed on computers. In this case, there will be digital files associated with the objects and these records should definitely be saved. Prime examples are the design files for *Prom Week*'s physical prototype. All the proprietary cards and character sheets were designed in Adobe Illustrator before being printed. These types of files should be included with other game documentation, and should be stored with other born-digital development documentation.

## Born Digital Content

Digital games are designed and programmed mostly through the use of other software programs. Most cultural software and digital game development documentation is born digital – being created, stored and represented through computational systems. This section highlights, in more depth, the types of digital content produced, and how that content is shared and stored during development. The first section, on game development files, describes the variety of digital files associated with a game and elaborates a brief taxonomy of file classes. The second section, Correspondence, covers email documentation and other means of communication between team members. The third section is devoted to document storage and organization during active development. Cloud services and source control software are the main storage methods for digital game assets. Each section and subsection provides general advice for game development process documentation and then focuses on specific issues derived from the analysis of the *Prom Week* project. We hope to illuminate the complexities facing archivists tasked with managing development documentation through advice paired with practical examples from our *Prom Week* analysis.

## Game Development Files

Academic game development processes produce a prodigious amount of born-digital files. These files can be organized based on three different production processes:

1. Software development files are those associated with a game's technical implementation.
2. Active research files are those associated with academic publications, commentary on research results, research data, or are themselves secondary research software.
3. Self-documentation files are those created by the development team to document the game and its process for press, conference presentation and collaborative development.

### *1. Software Development Files*

Software implementation is the process of translating a game's design into a playable computational object. In order to understand documentation associated with this process, we will first describe the basic technical structure of a computer game. This is important in disambiguating certain classes of files and showing how they relate to one another.

A digital game is usually composed of **source code**, **creative content**, and **external assets.**[1]

### *Source Code*

Source code documents are programming language text files that describe a game's logic, manage its presentation, and interface with a host platform's operating system and underlying hardware. However, just having the source code is not enough to run the game software. The source code must be translated from a human-readable programming language into a series of instructions understandable by a computer's processor. This process is called compilation, in which a second program, a compiler, reads the program source code and then organizes it into a file called an executable. The processor then executes the series of instructions in the executable and the result is a running version of the game software for a specified architecture or platform. Executables are also referred to as "binary executables" or just "binaries" because they contain binary representations of machine code instructions interpretable by a specific computer processor. Sometimes source code is compiled into intermediate representations that are then interpreted and run by other software. For example, all Java language based programs compile to Java bytecode, which is then usable on any computer running the Java Virtual Machine (JVM). Java bytecode is then portable, in the sense that any computer able to run the JVM (that is, any computer architecture for

1 Code is also creative, however we feel that this demarcation is sensible enough and that "not source code" is a bit too general.

which the JVM has been compiled) can also run Java bytecode.

Source code documents are the primary traces of the technical knowledge work for a digital game. Most of the innovation in programming and software engineering craft is contained in the source code files. Because it is very difficult or sometimes theoretically impossible to reverse a compilation process,[2] companies concerned with their intellectual property commercially release pre-compiled files when distributing a new project. This is what "closed source" means: the code is closed to inspection and modification as a side effect of compilation. Given that we are discussing academically produced games, there is an assumption that the source code will be open, findable and available to the archivist or future historical researcher.

If the source code documentation is available it will usually be stored together, typically in a collection of source folders. Source code, due to constant revision during development, is usually stored in source control repositories (as mentioned in the process section above). The benefit for archival ingestion is that source control repositories are usually accessible on the public Internet with the correct access credentials, or in read-only format if editorial access is not obtainable.[3] It is therefore possible to download all the assets needed to run a specific game, along with a record of all file modifications and team member contributions. Some software projects might separate the storage of source code and creative content, in which case it may not be possible to reconstruct the software.

Some management issues arise if the development team did not design their file storage for public access. We recommend that all projects plan for eventual public access at the beginning of a project as development teams may be less inclined to alter document storage during development. Direct access to an entire, functioning code-base is an organizational boon for digital collections and archives. Assuming that the archivist can recreate the hardware and software requirements of the development machine (either by purchasing the correct hardware or using a virtual machine to simulate it), it is more likely that they will be able to compile and run the game.[4]

### *Technical Documentation*

Technical documentation is a subset of source code documentation that communicates implementation decisions to other members of the team.[5] Sometimes the procedures for compilation and execution of a particular piece of software are complex, especially if there are a lot of external dependencies required (see external assets below). Teams engage in various levels of technical documentation, from comments written directly into source files to extensive external reference documents and wikis. It is important to keep these types of files consolidated with the source documentation to ensure future understanding of a system's technical architecture. It is here, and not in academic publications, that the working system will be found and hopefully understood. Technical documentation is also likely to be stored in a source control system, as it needs to be modified alongside technical implementation.

2 The process of reversing a compilation is called decompilation and is highly dependent on the type and specification of the data.

3 Typically, downloading information from a repository that is read-only only means that the user cannot upload new files or change files in the source repository. After download, the local copy of the repository's information can be manipulated in whatever fashion desired.

4 Organizing the correct virtual environment, or finding the correct physical hardware, is one of the most difficult technical issues facing long term software preservation.

5 This class of documentation and the self-documentation for outside developers are closely related. The specific interest here is documentation for those on the immediate development team during the development process.

One important type of technical document is the configuration information of the programmers' development environments. Most source code is written with the help of specialized editing software known as integrated development environments (or IDEs). IDEs are usually specific to a programming language or development platform. One might have an IDE for the C++ language (like Microsoft's Visual Studio) or for Flash-based game development (like Adobe Flash Builder). In either case there will be files that configure these environments for a development project. Because these files are sometimes developer specific they will not be included in source control repositories. In fact, any developer specific system settings or configuration files of any kind are usually explicitly barred from shared code bases to avoid confusion. While the development team in general might not have interest in a specific team member's process, future historians might. Obtaining these configuration files is probably beyond the scope of most software preservation efforts. However, if individual developers stored them in a consistent and distinct fashion, they could be included somewhere in a repository or archive.

Many projects also rely on external bug-tracking software to record software issues. These systems allow teams to assign problems to specific team members, and record when and how the issues were resolved. Some source control systems also interface with bug tracking software, meaning that they will be integrated into the project documentation.


Early art asset draft for Prom Week background settings.

### *Creative Content*

Creative content includes audio, visual or other structured representation files used by the compiled source code in the execution of the game. Sometimes creative content is included in the executable file, and at other times it remains separate (and is subsequently loaded and referenced by the executing program). These assets are distinct from source code in

production method, size and consistency of format. They are produced by a wide variety of third party software, including audio creation, image manipulation and 3D modeling programs. The assets produced are static configurations of data in specific formats or file types. For instance, an artist might export a photo from Adobe Photoshop into a JPEG file, a common image format. Creative content is typically much larger than source code files since visual and audio representations require significantly more space to encode their information. Additionally, these files are also referred to as static assets. They are not executed or compiled, and remained fixed after they are created. Because creative content files do not change frequently (if finalized), their revisions may be left out of version control repositories. Creative content files can be quite large and inflate the size of a version control repository, especially if many revisions to a large file need to be stored. Artists may also have many working revisions of an asset that are not related in a successive way (i.e., trying out different styles for an illustration). These are not as amenable to source control because stylistic revisions are not all used in the final game and take up significant space.

Creative content comes in a large variety of formats, some standard and some custom to specific development efforts. Reading these files in the future is going to be an issue unless there is some consistent way to identify which program (and more specifically which version of that program) created a specific asset. The *Prom Week* discussion below will provide a detailed look at the issues with creative content file types.

### *External Assets*

External assets are any files referenced or used by the executable that are not produced by the development team. Sometimes source code will be compiled into smaller chunks that share common functionality; these external libraries of code are then linked to the executable for use when the program is running. Programmers borrow or license other software libraries if they do not want to redesign functionality already created by other programmers. Teams may also license creative content, but it is usually integrated with other team-produced work. External assets also include software not written by the development team but necessary for correct operation of the game. A game may need to interface or communicate with a separate server to function properly (if it is multiplayer) or rely on a software platform for its interface to the computer. Any functionality provided by a game but not written by the development team may also contain source code or even further software dependencies for its operation. As a result of these nested dependencies it may not be possible to access all of a game's functionality even if you possess all of the files created by the development team.

### *Game Engines*

Game engines are collections of external libraries or secondary programs that generalize common tasks associated with game creation. Many game developers license game engines and then use them to construct a game. Game engines vary greatly in technical sophistication, programming requirements, and integration with other types of game development files. Some engines, like Unity, incorporate code and creative content into a single, graphical interface driven program. Others are just collections of compiled or open-source libraries that handle common game related tasks, like drawing objects to the screen or managing physical simulations. Game engines ideally function as abstraction layers between the creative content of a game (that dictates how it looks and sounds) and the program code (that organizes game play logic and manages graphical and auditory presentation.)

### *Licensing*

Many external software libraries include specific software licenses and restrictions. Open source software and libraries are not necessarily free of commercial restrictions even if all source code is available for use and modification. Licenses may dictate terms for commercial use and dissemination of included source code or other resources or, in the case of "free" software licenses, require that any code making use of licensed code itself be made freely available. Licenses can be included on a per file basis for external source code files, or organized to cover an entire toolset, secondary software tool, or larger group of source code files. In archiving game development documentation, there is a risk of breaking licensing conditions if materials are made freely available. Game and software development is built on the sharing of code, and sometimes even the development team (especially of non-commercial titles) may be unaware of licensing restrictions.

### *2. Research Files*

Research documentation is any pre-publication commentary and data generated by the development team. Academic game development is unusual in that the development team is simultaneously building a piece of software and a research agenda. The game system is a research output, and the details of its design and implementation are subject to publication as it is being constructed. Additionally, the functioning game may be collecting data from the players using it, or generating diagnostic information for researchers. Research data will probably be stored in some consistent location for access by the research team. Researchers may also independently store publication related materials and personal research notes. It is not unusual for a subset of the development team to be publishing about their specific contributions to a larger system or project. This means that documentation for specific aspects of the research may not be in a centralized location or shared with everyone on the project. Publication drafts and research notes are essential to understanding the process of research and development and should be saved.

Research data presents unique challenges if it is produced in significant quantities. Researchers generally use a subset of data for conclusions or trend analysis. These trends and conclusions are based on the data but do not rely on it after publication or the conclusion of research. The scientific contribution is usually the analysis of the data, not the data itself. Given the vast increase in available storage in recent years, it is probably possible to save most of the data output by an academic game. However, this might require removing data from more specialized locations, like databases, for preservation. Research data has its own format and long-term retention issues, though hopefully the information is available in a standard or understandable form. If the data is already reviewed and unlikely to be used again, it is a lower priority for preservation. On another note, if a game is still active and generating results for a research team even after its major documentation is stored in an archive, there is no clear recommendation for what to do. One could wait until the research project is completed and data collection has ended, or, if it will go on indefinitely, set up some procedure for ingesting older data into the archive after it is finished with active use. Methods for dealing with continuing data streams from ongoing projects need more attention and we recommend this as an area for further study.

### *3. Self-Documentation*

Development teams create promotional materials for press and conference appearances. Screenshots and gameplay videos document a game and provide a preview of the game play experience. When source code is unavailable, unrecoverable or unable to compile, promotional materials might provide the only understanding of how a game functioned and what it was like to play. Many academic games may not ever document themselves in this fashion, and it should probably be standard practice to produce secondary video

documentation of a system functionality and user interaction. Press materials are unlikely unless the game receives distinction for its gameplay or research outputs. In that case saving this information is vital to understanding the game's place in the larger community.

Development efforts can also record development history through blogs or other social media outlets. Designer notes and feedback from players may also be communicated in online forums and social media. Saving information posted to third-party social media sources is difficult and it is unlikely that the development team would save socially oriented online missives. Development blogs, however, are important records of development and should be included with any cultural software collection.

Documentation may also be provided for outside developers if the system or codebase is robust enough for external use. This documentation is different from internal source documentation in that it is for the benefit of those outside the main development process. In an academic research game context, this documentation might be associated with content creation tools, related to application programming interfaces (APIs), or describe non-obvious installation procedures. Documentation provided for users and developers unfamiliar with the development process is valuable to a preservation and archival effort. Any document designed to share knowledge about a system for a potential developer is probably more detailed than any research publication.

## Prom Week Files

The descriptions above represent a basic and ideal organization of development documentation. In actual game development practice document organization is more confusing and indistinct. Below is a description of *Prom Week*'s documentary organization to illustrate how an understanding of game development documentation can help in the appraisal process. We also highlight numerous issues and inconsistencies revealed by the *Prom Week* materials. All documentation mentioned is obtained from the *Prom Week* version control repository, shared document folders on the cloud, and the developers themselves.

### *Organization*

*Prom Week*'s version control repository held most of the project's finalized documentation. Cloud services hosted additional files related to demonstrations, research publications, and individual development efforts. Both locations contained a lot of unused or preliminary documentation, including of abandoned features and revisions. Shared cloud documentation had no discernible organization. Developers from the team helped establish the pertinence of some cloud files, others remained a mystery, lost in the development shuffle. On the following page are the categories of files found in *Prom Week*'s documentation, along with their most common locations (version control, cloud services, or personal storage).

### *Prom Week Document Classes*

Research Documentation (cloud)

- Publication
  - » Outlines
  - » Shared notes
  - » Revised publications
  - » Finalized publications
  - » Rejected publications
  - » Submissions
  - » Abstracts
  - » Figures
  - » Templates
- Physical Presentations
  - » Posters
  - » Poster templates
  - » Presentations
  - » Conference ephemera
- Assignments for undergraduate researchers

Game Development Documentation

- Internal data analysis (cloud)
- Task lists (cloud)
- Development planning documentation (cloud)
- Digital prototypes (cloud, personal)
- Test applications for various game components (cloud)
- User interface mock ups (cloud, version control)
- Creative content (cloud, version control)
  - » Background images
  - » Character art
  - » Music files
  - » Video files
- Back ups (cloud)
- Implementation documentation (version control)
  - » Application project files for integrated development environments (IDEs)
  - » Application specific files
  - » Structured data files
  - » Shell scripting files
  - » System configuration files
  - » IDE configuration files
  - » Source code
    - * Game processes
    - * Data processing
    - * Asset management
    - * Data structures
    - * User Interface
    - * Database communication

Self Documentation (cloud)

- Screenshots
- Gameplay videos
- Presentation videos
- Web Site
  - » Web browser icons
  - » Implementation documents

Our familiarity with Flash and ActionScript 3.0 development practice formed the basis of the file categorization above. We are aware that this level of documentary scrutiny is beyond the archival faculties of most institutions. The breakdown is intended to help in the quick appraisal of digital files.

### *Prom Week File Types*

Finding software to interpret all the different file types was another problem for *Prom Week*'s appraisal. We ended up identifying 60 different file extensions associated with over 30 different programs. The list is provided below, followed by a discussion of some particularly prickly file types.

The table below provides the file extension, a description of its content, the program required to create it, and a note about its readability as plain text. This last designation is important in that any file readable as plain text is more amenable to future interpretation by researchers. The criterion for plain text readability is whether a file type's function is determinable through examination by a simple text editor. Some of the files readable in plain text may still be unrepresentable, especially if the programs they depend on are no longer available. For instance, a Scalable Vector Graphics file is simply a giant list of coordinates and vector transformations. To see the image described in the file, one would need to write (or locate) a piece of software to render the vector image it describes.

| File Extension | File Description | Dependent Program | Plain Text? |
|---|---|---|---|
| `.7z` | 7-Zip Archive | 7-Zip | N |
| `.ai` | Adobe Illustrator project | Adobe Illustrator | N |
| `.as` | Actionscript 3.0 source | Any Actionscript compatible IDE, any text editor | Y |
| `.asproj` | Flash Develop Actionscript 3.0 project | Flash Develop | Y |
| `.au` | Audacity block | Audacity audio software | N |
| `.aup` | Audacity project | -- | N |
| `.aup.bak` | Audacity project backup | -- | N |
| `.avi` | Audio Video Interleave | Any video player or video editing software | N |
| `.bak` | General backup | Dependent on backup type | ? |
| `.bat` | Microsoft Windows batch process | Microsoft Windows operating system | Y |
| `.camproj` | Camtasia Studio project | Camtasia Studio 7.0 | Y |
| `.camrec` | Camtasia Studio recording | Camtasia Studio 7.0 | N |
| `.css` | Cascading style sheet | Any web browser, text editor | Y |
| `.csv` | Comma separated values | Any spreadsheet, text editor | Y |
| `.dll` | Microsoft Windows Dynamic-link library | Microsoft Windows Operating System | N |
| `.doc` | Microsoft Word document | Microsoft Word (pre 2007) | Y |
| `.docx` | Microsoft Word document, object oriented XML | Microsoft Word (2007 or later) | Y |
| `.dropbox` | Dropbox configuration | Dropbox | Y |
| `.fla` | Adobe Flash project | Adobe Flash | N |

| File Extension | File Description | Dependent Program | Plain Text? |
|---|---|---|---|
| .fxp | Adobe Flash Builder Flex project file | Adobe Flash Builder | Y |
| .gif | Graphics Interchange Format | Any image software | N |
| .html | Hypertext Markup Language (XHTML, HTML 4.01, 5.0) | Any web browser, text editor | Y |
| .java | Java source | Any text editor | Y |
| .jpeg | Joint Expert Group image | Any image software | N |
| .jpg | Same as .jpeg | -- | -- |
| .js | Javascript source | Any web browser, text editor | Y |
| .lel | Windows log in screen editor | Deviant Art community application (no longer available) | N |
| .m4v | Apple Video container | Any video player | N |
| .mov | Quicktime Video | Quicktime | N |
| .mp3 | MPEG-1 or MPEG-2 Audio Layer III digital audio | Any music player | N |
| .mp4 | MPEG-4 Part 14 multimedia format | Any video player | N |
| .mxml | Adobe Flex meta XML | Adobe Flash Builder, any text editor | Y |
| .odt | OpenDocument text | OpenOffice Writer | Y |
| .odp | OpenDocument presentation | OpenOffice Impress | N |
| .old | Backup | Dependent on backup type | ? |
| .pages | Apple Pages document | Apple Pages | Y |
| .pbm | Portable bitmap | Netpbm | N |
| .pdf | Portable Document Format | Any PDF viewer | N |
| .php | PHP source | Any text editor | Y |
| .png | Portable Network Graphics | Any image software | N |
| .potx | Microsoft Powerpoint Template | Microsoft Powerpoint (2007 or later) | N |
| .ppt | Microsoft Powerpoint presentation | Microsoft Powerpoint (pre 2007) | N |
| .pptx | Microsoft Powerpoint presentation, object oriented XML | Microsoft Powerpoint (2007 or later) | N |
| .psd | Adobe Photoshop project | Adobe Photoshop | N |

| File Extension | File Description | Dependent Program | Plain Text? |
|---|---|---|---|
| .pyd | Windows Dynamic-link library written in Python | Microsoft Windows operating system | N |
| .rtf | Rich Text Format | Any text editor | Y |
| .sql | Structured Query Language source | Any text editor | Y |
| .svg | Scalable Vector Graphics | Any image software | Y |
| .swc | Compiled Shockwave Flash file | Adobe Flash / Flash Builder | N |
| .swf | Shockwave Flash file | Adobe Flash / Flash web browser plugin | N |
| .txt | Standard ASCII text | Any text editor | Y |
| .vpk | VUE package file | Tufts Visual Understanding Environment | N |
| .vue | VUE concept map | Tufts Visual Understanding Environment | N |
| .wav | Waveform audio file | Any audio player | N |
| .xcf | GIMP project | GNU Image Manipulation Program | N |
| .xlsx | Microsoft Excel spreadsheet, object oriented XML | Microsoft Excel (2007 or later) | Y |
| .xml | eXtensible Markup Language | Any text editor | Y |
| .xmpses | Adobe Premiere Elements DVD marker | Adobe Premiere | N |
| .zip | Zip file archive | Any extraction program | N |
| No extension | File folder / directory | Dependent on operating system | ? |

### *Versions*

Many of the file types listed above do not have specific versions of dependent software listed. This is due to multiple programs using the same file extension through multiple upgrade cycles. For instance, Adobe Creative Suite programs, like Photoshop and Illustrator, share the same extensions for most of their versions. Some features in older files are not representable in newer versions of the software and vice-versa. This creates an issue when you load an older file in a newer program; it attempts to change the file into a newer format, making it unreadable to the program that created the original file. Determining the versions of a specific file type are not straightforward, and though our analysis used multiple file identification tools, none were particularly suited to the diversity of game development documentation.[6]

### *Obscurity*

Some other file types are so obscure that they may remain unidentified. Two examples on the above list are .vue and .lel files. Without extensive research (something far beyond usual archival considerations) it was not possible to determine what the files were or what program created them. Starting with the .vue file, which is a Visual Understanding Environment file

6 Identification programs include: DROID (http://digital-preservation.github.io/droid/), JHOVE (http://jhove.sourceforge.net/) and JHOVE2 (https://bitbucket.org/jhove2/main/wiki/Home). Bit Curator from University of Maryland might be more useful but was released after our investigation concluded (http://www.bitcurator.net).

for a Tufts mind-mapping software program, our identification efforts met with significant difficulties. All common information available pointed to the .vue file being a 3D geometry file, which made no sense in the context of *Prom Week* (a 2D game). After examining the header information of a totally different file type, a .vpk file, we determined that the .vpk was an archive of .vue files and that it was associated with Tufts University. Only after making the Tufts connection could we determine the origin of the file extension.

The .lel file was even more obscure, and after significant web investigation, our best guess is that it was associated with a Windows operating system modification program. The online community DeviantArt provides user created programs to modify operating system themes and aesthetics. A program that modifies the Windows 7 login screen seems to have accepted .lel files. This is no longer verifiable, however, because there is no longer any trace of the program online. All links to available downloads no longer work. Without an explicit record of the types of programs used in development, it may be impossible to determine the content of future files.

## Email Correspondence

Some modern development correspondence is in the form of email between team members.[7] Project teams organize email lists based on function, with general correspondence, content creation and technical implementation sometimes occupying individual lists. Academic researchers usually use official institutional email accounts and mailing lists throughout the course of project. This makes it possible to acquire a downloaded version of a project's email archive. Any email sent to a list should be available, with the obvious exception of emails sent between team members individually and not to a list. Email archives come in a variety of formats. One of the most popular is the mbox format family originally developed for Unix. Other archives may available as compressed collections of email text and identifying folders, or as GNU mailman archives. Given the existence of email parsing programs, as long as the email text is in a recognizable format or structure it should be possible for future researchers to interpret it. Email correspondence helps explain project process and structure, and also cements the development timeline. Researchers are sometimes fuzzy on the exact timing and order of events; the email record can emphatically clear that up.

Any large collections of personal communication invite concern over user privacy. Given that these email lists are about specific research and development issues, there is probably no improper information present. List members and maintainers should be contacted regarding potentially dubious subjects before making an email archive publically available.

### *Prom Week Email Correspondence*

*Prom Week*'s correspondence occurred on 7 separate internal mailing lists all hosted on UCSC mail servers. Lists organized around specific team activities:

- Altprom-artgroup
    - All work on in-game art, sound and other creative content
- Altprom-authors
    - All authors writing *Prom Week* in-game conversation text
- Altprom-closers

7 Teams may also correspond through bug-tracking, version control or development support systems, in source code comments, and / or in comments to online-shared documentation.

    - List devoted to a smaller group of developers focused on finishing the project after undergraduates left the project
- Altprom-commits
    - Emails sent by the source control server describing recent code changes
- Altprom-general
    - List for everyone involved in the project, provided general announcements and information
- Altprom-group
    - Probably a smaller sub group of core developers (development team could not remember what this list was for)
- Altprom-tigers
    - Initial *Prom Week* email list, consisted of core development team only

All email lists are prefaced with "Altprom" (a name derived from an early version of the game based on an "alternative" prom theme). A *Prom Week* lead developer provided us with a full email dump of development correspondence. The records arrived as a zipped mailman archive, and were parsed for basic information (like identifying the purpose of each list) with the Stanford developed MUSE email analysis tool.[8] Some issues arose in that the mailman archive did not record who sent specific emails. As a result, all emails are addressed as coming from the server itself. Email helped to identify a timeline for key development events, but it was sometimes impossible to know who sent them. Luckily, most list members had identifying footer information automatically attached that aided in figuring out email provenance.

The *Prom Week* project took years, and the members of each list changed over the course of that period. The emails did account for everyone involved in development, and even helped core developers to remember when certain people joined and left the team. Due to the length of the project, some core development members were unsure of what each email list actually discussed. Some smaller lists were created for specific, short-term purposes and then changed focus or were abandoned.

8 MUSE is software developed by Stanford's MobiSocial Computing Lab. MUSE Project link: http://mobisocial.stanford.edu/muse/

## Cloud Services

Cloud services are any online document collaboration or backup system controlled or managed by an outside corporation. Common examples include Google Drive and Docs, Box.net and Dropbox. These services are convenient for sharing files amongst team members because they ensure a consistent and safe location for documentation. Although the services are not uniform in functionality, they all share common shortcomings that can create problems for digital archivists and long-term preservation. The types of files stored on cloud services tend to be organizational documents, demonstration files, and non-finalized creative content. All source code and active technical implementation files are found in version control, discussed in the next section. Removing information from these services is paramount as they are not designed for long-term preservation or storage and are subject to market forces.

### *Access Restrictions*

Services require users to have accounts for upload and collaboration. Additionally, once an account is set up it needs to be granted access to specific files or directories. Institutions, therefore, need to maintain an active account on each service used by the development team, and then rely on team members to provide access. Hopefully, all development files are organized in a coherent fashion, otherwise document sharing might prove onerous for team members needing to individually share files, or locate them across multiple locations in a shared directory.

### *Migration*

Once files are located and accessible, it can be problematic to move them off of a service. One issue is ensuring that the file tree hierarchy and folder organization remain intact after a download. Maintaining file organization is important for understanding development process and for locating specific files. Many services provide robust features for search and organization based on tags, keywords, and other service specific functionality. In many cases these organizational structures will be lost when files are migrated. Additionally, most services will record file creation and last-modified information through timestamps embedded in a file's metadata. Because downloaded files may represent new creations from the perspective of the receiving operating system, this creation and modification information can be lost in transfer. Migration presents serious problems to file metadata integrity and should be handled with care when encountering files stored on cloud services.

### *Revision History*

Many cloud services provide some form of file version and revision tracking. While this functionality is convenient for users of a service (in case they make an accidental change or mistakenly delete a file) the information associated with revisions may not be accessible after removal from the service. In order to save space, it is common for cloud service providers to keep only a certain number of revisions or to slowly delete revisions for files that have not changed over long periods. It may be possible to pay for service enhancements to keep revision information permanently, but accessing and downloading previous revisions is generally onerous and technically challenging (see next section on APIs).

### *Application Programming Interfaces (APIs)*

APIs allow developers to write software that can interact with a service. The methods used to enable system interoperability are beyond the scope of this report. However, we mention them here because certain file attributes, metadata, and revision history information are sometimes only available to programmers. This needs to be considered in any archival plan dealing with cloud services. Someone familiar with software development should evaluate information available through APIs, otherwise pertinent information may be lost after data is transferred off of an online service.

### *Dropbox and Prom Week*

Dropbox is an online storage service that allows users to sync a folder on their local computer to a shared network folder. This is convenient because it allows users to passively backup files because they do not have to manage file updates or uploads to the service. The *Prom Week* team used a shared Dropbox folder for documentation not already stored in source control. Every team member had access to this folder, and that is reflected in its haphazard structure. There is no consistent scheme for file folder names, each being named for a specific task, like a conference publication, or a specific person ("Ben's Stuff"). The Dropbox account contains over 4GB of documentation and over 4000 individual files. The documents cover every category listed in previous sections, although any source code is usually from demonstration versions of the game or unrelated programmatic tasks.

We had to create and use a Dropbox account to view the files. A project lead gave us full access to the hosted folder. This presented immediate problems because Dropbox automatically syncs files to a local folder. As a result, all the files we downloaded had incorrect creation and last-modified dates. Our local operating system had treated all the files as new and dated them accordingly. If we logged into our account with a web browser, the files listed there had the correct creation and modification information. The system also provided information unobtainable from the files themselves. Dropbox recorded the name of the last team member to modify a file, and also provided access to revision histories. This information was downloadable through the Dropbox APIs, but required us to write Javascript and Python code to access it. The scripts we wrote showed that such information could be removed from the service, but we do not have recommendations for the best way to make use of it, nor attach it to specific files. Extensive file revision functionality is available through Dropbox's Packrat service. The *Prom Week* team did not use that added (paid) feature and so their files have no accessible revision histories.

### *Google Drive and Prom Week*

Google Drive is the umbrella name for Google's document manipulation and storage services. Google Drive can create documents that are manipulated by Google Apps, a basic set of office document tools for word processing, slide presentations, and spreadsheet manipulation. Unlike Dropbox, Drive stores all files exclusively on Google's servers, and users must have Google accounts to access them.[9] Multiple people can manipulate Drive created documents at the same time, making them good for collaborative efforts. The *Prom Week* team stored around 40 files on Google Drive, and they mainly related to documentation of the development process. Things like task lists, prospective features, and design documents are among the most prevalent files.

A project lead shared their Google Drive folder with us after we set up an account on the service. All files were stored in a single folder. Google Drive's interface allows you to share individual files located anywhere on the service, making it potentially problematic if a number of development files are located outside of a shared folder. Google documents also have no fixed logical form. When downloading a document from the service, the online file is converted into a variety of formats based on user request. Therefore, migration and information loss must be immediately considered when dealing with these files. Google also provides an extensive API for obtaining information about files and is thorough in documenting the metadata available through the service. Google documents also record document histories and revisions, but they are sometimes overly specific. Additionally, Google Drive culls old revisions at regular intervals if there is not recent activity. If a file is being heavily edited, however, the revision history will list every minute edit by each individual. Clarifying a consistent policy for revision tracking will be paramount in dealing with a potential deluge of revision files in Google cloud services.

9 Google has added functionality similar to Dropbox's folder syncing but it requires additional setup and is not a default feature.

For our purposes, we downloaded each file in every available format. A Google word processing document, for example, is exportable to:

- Portable Document Format (.pdf)
- Plain text (.txt)
- Rich Text Format (.rtf)
- Microsoft Office (.docx)
- OpenOffice (.odt)
- Web page (zipped .html archive)

Exporting eliminated all revision history and file modification information. As with Dropbox, we were able to recover that information from Google through their JavaScript API, but have not solved what to do with it nor what form it should take.

## Version Control Repositories

As discussed in "The Process of Academic Game Development" above, version control's goal is to manage document collaboration between developers. By recording and organizing revisions in source code and other development files, version control insures that only the most recent versions of a file are being edited. Or, if there are multiple versions of the same file, version control keeps them separated, one unable to interfere with the other. In order to keep track of files, version control software creates a file repository, a dedicated location for files used on a project. The repository may be centralized, in that all files are managed in one single location (usually a network-connected server), or distributed, in that each developer has their own private repository that communicates with others. Centralized repositories maintain the most up to date versions of development files, and track any changes made by any member of the team. Distributed repositories each keep their own records and make sure they are synchronized with other repositories also containing the same project. Version control is used to house the technical implementation files for a software project. Most of the files in source control are source code files, along with any creative and external assets needed for a piece of software to compile and run. Version control can manage any type of file, but is specially suited and designed for the needs of code creation and development. Version control is where the files most important to understanding a game's technical implementation and architecture are most likely to be found. Archival issues associated with version control are similar to those found with cloud services. However, because revision tracking is paramount for these systems it is much easier to ascertain whether all potential versions of a file are available. Additionally, most version control systems are designed by programmers and are open-source, so they do not require the use of third party servers to function.

### *Access*

Version control systems for academic projects at universities are generally set up inside a university network on dedicated servers. Some other academic institutions, like smaller colleges or design schools, might elect to have external web or cloud-based services handle version control. In any case, access credentials to each project are required to check code in and out of the system. This may present a problem if the server is not accessible outside a university network, or if a given external service is unavailable. Given that most academic research projects benefit from sharing and dissemination, a public, read-only version of the repository is usually available. This is assuming that the development team or version control administrator can provide an archivist with a user account and access, or that access is free

to anyone with an Internet connection. External version control servers may have associated fees for different levels of functionality and should be evaluated as cloud services, with all the requisite migration issues. When a third party manages version control, credentials for the third party service and the individual repository need to be acquired.

### *Migration*

Version control systems are designed for file sharing through constant upload and download operations. It is usually possible to download all recent files from a project without much trouble. Problems arise in that when using centralized version control systems old revisions are stored in the central repository. To access the older versions, one must request them from the server using the version control software. The software will then download the files and make changes accordingly. Distributed systems mitigate this issue somewhat, because all revisions are stored locally before being forwarded to another repository. However, different distributed repositories may be in different states at different times. Additionally, in distributed systems developers choose which changes to propagate to others. In some cases entire sub-systems or segments of development documentation may never be shared.

Version control repositories are themselves groups of files. They have their own formats, and they allow the software to manage file changes and tag certain files and directories for revision. To save all version control documentation, it is necessary to save the files in the repository as well as the repository itself. Many systems allow for the cloning of an entire repository. This functionality is necessary for developers wanting to migrate their own repositories to different machines. The long-term issue is that version control software is subject to the same preservation issues associated with any software. If there is not a suitable emulated environment or specific system configuration information available, version control software will not run, effectively preventing full usage of repository files. Take note of the type of version control system in use, including its dependent run-time context and its software version. Version control software is subject to updates that may render older versions of a repository inoperable. Knowing what software created a repository is a good way to ensure it will be accessible in the future.

### *Prom Week Version Control*

*Prom Week* used version control to manage all source code and technical implementation documentation. The project used Subversion, a centralized version control system. Subversion, abbreviated as "svn", is the default version control software for the School of Engineering at UC Santa Cruz. A shared server hosted all student projects and managed access through standard university accounts. Our project team included a CS affiliated graduate student and an associate professor; therefore we did not need to obtain university credentials. The *Prom Week* project lead granted us access to the source repository and we had no trouble copying all development and repository files. A slight note, during the course of our investigation the svn server changed domains. As a result most of the development team did not know exactly where their files were for a brief period. Luckily because *Prom Week* is still an active research project, team members almost immediately remedied the situation. An older project or one without recent activity might have been lost entirely.

*Prom Week*'s source files include numerous versions of the game, along with demonstration versions and related side projects. This makes finding the actual files responsible for the final game rather difficult, but provides a rich set of materials of potential interest to future researchers. Additionally, external dependencies are mixed in with source files, making it sometimes hard to identify files created by the research team. Many research dead ends and half-implemented features are present, still lingering in sub-directories, which again

compounds the trouble of finding a singular set of development documentation while simultaneously offering important future insights. The repository is a representation of the hectic, dynamic environment of research software development. *Prom Week*'s documentary complexity is not unique, after years of development most software projects will contain a significant amount of dead code and abandoned effort. In appraisal it is most useful to just save everything in the repository, since it will not be immediately obvious which files are important to the project or future researchers.

## Repository Storage and Preservation Strategy

Digital repository structures, processes and retention strategies can vary greatly between different institutions. This section covers two examples of digital repository storage of game development documentation at UC Santa Cruz. *Prom Week*'s storage into UCSC digital repository is described, as is the storage of games from UCSC's Foundations of Game Design undergraduate course. The methods briefly described below are likely transferable to any institution with some form of digital repository.

### *Prom Week Documentary Storage*

UCSC (as well as the larger University of California library system) provide digital repository storage services through the Merritt online digital repository. Files stored for long-term preservation are indexed and searchable based on title-level descriptors (like title and creator). Larger groups of files are compressed and uploaded to the service where content listings of individual files are automatically created. To upload *Prom Week*, its documentation was broken up into 24 zip archives containing all digital documentation for the project and recordings of the interviews conducted for this report.

The storage process involved:

**1. Organizing all documentation into coherent chunks for compression**

*Prom Week*'s documentation was grouped (by the development team) according to online services. That is, the original born-digital file organization consisted of:

- Shared folders on Dropbox and Google Drive
- Email archives for development mailing lists
- Source code and finalized creative content in Subversion version control

Individual interviews with the development team were stored, with topical indexes, in separate compressed archives. All data stored on online services was left with its original file organization and hierarchy, even if that made organization less clear.

**2. Creating a file manifest for the compressed documentation**

The manifest is a spreadsheet describing the title (general description), original creation date and creator for each archive, in addition to a hashed checksum for file verification.[10] Descriptions are necessary for search and indexing in the digital repository. Titles included an overview of the contents of each archive and (when applicable) the version of the storage software used.

**3. Uploading finalized compressed content to the repository**

After validating the files according to the generated checksums, each file is paired with its descriptive information and available for download from the repository.

*Prom Week*'s combined documentation included:

- A combined archive of email archives in mbox format for all project mailing lists
- Contents of the development team's shared Dropbox folder
- Contents of the development team's shared Google Drive folder in multiple formats[11]

10 The md5deep hashing tool was used to generate MD5 and SHA-256 hashes for each archive file.

11 Google Drive allows for the export of text files into five different formats. An archive of each available format (plain text, PDF, Microsoft Office post-2007 [.docx], OpenOffice [.odf], and HTML) was uploaded.

- 13 development team interviews
- An archive of the online demonstrations and publications (development blog) of the *Prom Week* development team as collected by Archive-It.Org's web crawler[12]
- The most current version of the source code and creative content stored in version control
- A plain-text dump of the contents of the version control repository
- A copy of the entire version control repository, including all code and content revisions

### *Undergraduate Documentary Storage*

*Foundations of Interactive Game Design* (CMPS 80K) is an introductory game design course offered every year by the Computer Science department at UC Santa Cruz. The course is taught either by a professional game designer or a member of the Games and Playable Media group. Students in teams of two or three learn how to prototype and create basic computer games through the use of game development tools that do not require programming knowledge. Each year over 80 original games are created. The output of the 2014 course is the first to be digitally preserved by the university.

Student produced games were digitally stored on UCSC's eScholarship platform. Originally intended for academic research publications, the service also allows for the upload and storage of arbitrary data in compressed archives. The platform is not intended for software development storage, but it allowed for easy form-based submission of content that did not require the more extensive processes associated with the University of California's Merritt repository. eScholarship provided the ability to upload arbitrary blobs of data, thereby making an ad-hoc process possible for varied collections of game development data.

Teaching assistants for the course required students submitting their final games to describe them according to specific criteria, using controlled vocabulary tags to aid in future search and discovery. Tags were specific to the concerns of future game design students, so information about the genre, play style and technical content were included. The games and user manuals were then compressed into archive files and uploaded using a simple web form. Although the course does provide awards and recognition for the best games produced each year, it was decided that every game, regardless of quality, should be preserved for the benefit of future students. For the initial test process, only the final course output was saved. In the future there are plans to extend this process to cover all class output, including digital and physical prototypes, and initial design documents.

12 Archive-It.Org's tools output data in the WARC archival format. Due to the less common nature of this extension, we extracted the WARC contents using the 7Zip compression tool and included both the extracted contents and the WARC archive in our compressed file.

# Recommendations

These recommendations are derived from our analysis of the development of UC Santa Cruz's online Flash game *Prom Week*. We conducted a full archival appraisal of the software, interviewed the researchers, and aggregated all the documentation we could find on the project. During that process, it became apparent that there were measures most institutions could take to alleviate certain issues we encountered during the appraisal. Most of the recommendations are directed at academic researchers, and focus on making them more aware of the archival issues associated with modern software development methods.[1]

## 1. Establish a laboratory- or department-wide data management policy that outlines how data will be collected, organized, described and archived, according to the standards of the intended institutional or discipline-specific data repository.

**Addressed to:** Laboratory administration and researchers, university administration, in consultation with repository managers.

The California Digital Library (CDL) provides a data management tool (DMPTool, https://dmp.cdlib.org/) designed to help researchers create custom data management plans for their research projects. Additionally, many granting agencies provide guidelines and sample data management documentation for applicants.[2] These resources may be used to help tailor more local, intra-institutional data management strategies. Uniform data management policies can help insure that significant research effort and creative and scientific outputs are saved for future researchers, scholars and other interested parties.

## 2. Produce data and documentation in formats that are easily archivable or can be converted to easily archivable formats.

**Addressed to:** Laboratory researchers, in consultation with repository managers.

Archivable formats are those amenable to long-term preservation and storage. Any format with an open and available specification is considered to be better than closed and proprietary formats. Open specifications, if they are sufficiently detailed, ideally allow for the engineering of new software to interpret a format. Formats linked to specific commercial software and that are not extensively documented should be avoided when possible.

1 The format of these recommendations is influenced by the ACLS Commission on Cyberinfrastructure for the Humanities and Social Sciences Report: http://www.acls.org/cyberinfrastructure/OurCulturalCommonwealth.pdf.

2 The CDL maintains a listing of data management plan templates for many granting agencies and institutions at: https://dmptool.org/guidance.

### 3. Store all relevant development documentation in as few separate locations as possible. Create a manifest or directory that describes and provides access to all project documentation.

**Addressed to:** Laboratory researchers

Recommendation (2) and (3) together mean having all development work and documentation inside source control[3], or inside shared folders. Avoid singular documents outside project folders; anything that is hard to track down is probably at risk.

### 4. If possible, ensure that your source repository is available via the open Internet in read-only form.

**Addressed to:** Laboratory administrators and researchers

If you are working on an academic project there are potentially many concerns with sharing results and process documentation. These range from competition with other labs to issues with providing support to old, messy or half completed research software, among others. When openly sharing results is not ideal or possible, there are still other available avenues for long-term storage, including dark archives (with specific access restrictions and retention guidelines). If your results are restricted by agreements with outside parties such as sponsors, explore how to make the results openly available as soon as possible, perhaps by negotiating an agreement that will make the data available on a specified schedule. If possible, all code and data should be available for read-only download to help bolster against future loss.

### 5. Provide source control check in comments that are descriptive and can be understood by repository managers who were not involved in your project.

**Addressed to:** Laboratory researchers

Source control procedure usually requires the user to provide a description of the changes they are making to the repository. In practice this is beneficial to the development team because it makes it easy to see who made changes and why. Adopting a routine commentary structure would significantly help future researchers, if repository and archival staff can interpret these comments accurately. Some version control systems recommend specific commentary structures in their documentation.[4]

3 Source or version control repositories manage source code for teams of programmers and developers. Most university projects use these systems to manage code revisions and team collaboration. Some common systems are Git, Subversion (SVN), and Mercurial.

4 For example, the Git project documentation provides recommendations for comment style (http://git-scm.com/book/en/Distributed-Git-Contributing-to-a-Project).

Because version control comments are vital to most significant projects and software companies, many provide their own guidelines for code contribution and task description.[5]

### 6. Record software programs used in the creation of game files and assets (including format and version information).

**Addressed to:** Laboratory researchers

Keep track of the different types of software used to create game assets and documentation. This can simply be a manifest file listing the file extensions of specific assets and what software (and version) was used. It will be very helpful in the future to avoid file type confusion and afford future researchers a better chance of recovering game development information.

### 7. Conduct development correspondence on official group mailing lists as much as possible.

**Addressed to:** Laboratory researchers

The email record of a project is helpful in establishing the development timeline and individual involvement. Try to keep all development related correspondence in as few locations as possible (see recommendation 3). Even when it is necessary to maintain multiple mailing lists, best practice is to host them on a single academic server. This practice will better enable project and repository managers to gain access to all project correspondence. Also, avoid inclusion of personal data (social security numbers, credit card information, etc.) or discussions of a personal or sensitive nature in project correspondence. This will allow archivists more freedom in sharing development archives.

### 8. Assign responsibility for digital archives and institutional software development archives to a specific member on the document appraisal team. Ensure that this team member is well-versed in software development processes and documentation.

**Addressed to:** Digital preservation, digital repository and archival staff

Any game development project is a complex technical endeavor, and it behooves an ingesting institution to have individuals with development experience on an archival

5 For example, Linux kernel guidelines (https://www.kernel.org/doc/Documentation/SubmittingPatches) and Chromium browser guidelines (http://www.chromium.org/developers/contributing-code).

team. Software development is very similar to game development, so just a little experience with programming environments and development tools will go a long way in helping archive, appraise and retain development documentation.

### 9. Instate data management plans for all game design and development projects and courses.

**Addressed to:** University management, faculty, repository management staff

Many institutions pursuing academic game development research also tend to have undergraduates programs in the discipline. It is important to instill a sense of digital preservation technique and practice among the students in these programs.

### 10. Develop instruments (such as transfer and data deposit agreements) and policies for ingestion, description, and access with regard to institutional software development records.

**Addressed to:** Repository management staff

In most cases, an academic institution owns any intellectual property, software and documentation. Additional transfer documentation may be necessary to clarify agreements with students, faculty and staff participating in software development, or with third-party development partners. Data deposit forms should be available for the transfer process, as well as clear procedures for description and discovery metadata creation, and for access policies.

# Future Research

Based on our work with *Prom Week*'s documenation it would be immediately beneficial to pursue research in the following areas:

## 1. Methods for documentation extraction from cloud-based services

As mentioned above, most cloud-based services implement different file access, documentary metadata, and file versioning strategies. Research and further guidelines on a general set of strategies for institutional digital repositories interfacing with these services would be ideal.

## 2. More investigation of academic software creation practices

Our study focused exclusively on one mid-scale game development project. More case studies of other academic development projects, beyond their official publications, would likely reveal more issues and complexities. Document-focused case studies of digital humanities, or other cultural software projects in the social or natural sciences, would help extend digital archivists' knowledge, and let repositories better serve future projects' preservation needs.

## 3. Unified citation and description of software and computer games by libraries and archives

A clear set of citation and terminological guidelines for software and computer games, including cataloging guidelines, and metadata for description and discovery, is needed to ensure that such digital artifacts are locatable and accessible for future researchers.[1]

## 4. Development of tools for managing: (1) Cloud-based document extraction, (2) File type categorization, (3) Archival extensions for source and version control, and (4) Copyright- and rights-aware metadata

Future tools will help improve archival and organizational efforts for cultural software artifacts and their documentation. Besides extraction tools for cloud services, file type classification software for proprietary development files is needed. Current forensic file identification tools deal with more common formats, making such tools unsuited to the diversity of software development assets. Additionally, support for linking source control and archival repositories would also help automatically archive development projects. And lastly, work produced by academic researchers is subject to institution-specific copyright and licensing rights. Accounting for this information before ingestion into repositories would benefit archives, archivists, and collections managers.

1 The team formed for this NEH Digital Start Up, with an extended set of personnel, recently received an Institute for Museum and Library Services (IMLS) National Leadership grant to research metadata, terminology and citation for digital games. LG-06-13-0205-13.

# Conclusion

We have outlined an approach to aid understanding, appraisal and retention of documentation related to cultural software and its creative processes. Software creation can be a complex and confusing task for developers, and even more so for archival and collections staff untrained in software development methods and practices. As a first step towards alleviating some confusion in dealing with cultural software development records, we have illuminated three key areas inherent to the software development: process, context, and documentation. Each area presents a specific view of cultural software development, and helps to build an awareness of the challenges of analyzing development documentation.

We based our analysis on a detailed case study of Prom Week, a game (and, more broadly, a cultural software artifact) created by students and faculty at the UCSC Expressive Intelligence Studio (EIS) Computer Science laboratory. This approach was based on previous archival efforts into uncovering the *process* of technological and scientific creation. We drew inspiration from the JCAST report "Understanding Process and Progress: Documentation of the History of Post War Science and Technology in the United States," with its focus on preserving the history of scientific and technology progress; the Charles Babbage Institute's "The High Technology Company," with its case-based "documentary probes" of technical artifacts and their creative processes; and the NDIIPP "Preserving Virtual Worlds: Final report," the major previous study on the issues associated with long-term preservation of computer game software. Only through a specific, and inherently messy, journey through the process documentation of a complex software object were we able to find and document potential challenges for future archivists and scholars.

Over the course of our research, we interviewed numerous individuals associated with the Prom Week project, and got access to the a majority of the available project documentation. We then proceeded to analyze, categorize, and recommend appraisal strategies for all of its various types (down to individual file formats). The resulting mass of documentation, stored across version control systems, online cloud storage services (Google Docs and Dropbox), email servers, and personal websites, is now permanently stored in UC's Merritt Repository for future access and potential study. The approach we have outlined can be immediately applied to any other academically produced research game, and, we believe, to many other cultural software works produced inside and outside academy. This work immediately led to a scaled down approach to UCSC's undergraduate game design documentation, and is certainly applicable to a broad range of digital media works.

We also stress, throughout the report and in our recommendations, that most born-digital content stored on the cloud, or in unstable or non-permanent storage, is at risk of being lost. Perhaps the most significant takeaways from this project is the need to understand that process documentation should be saved, that cultural software documents can be organized and appraised, and that in saving said documentation one should find permanent and institutional long-term storage solutions. Many cultural institutions and archives can provide such services, and it is important to explain why cultural software documentation needs to be preserved, and to describe the form of the documentation to be stored. We have begun to communicate those needs in this report, and are counting on future archivists, scholars, and others concerned with historical cultural software to continue pushing for better methods and more direct case studies into all manor of software and other digital artifacts.

# Contributors

**Christy Caldwell**
***Librarian***
UC Santa Cruz

Christy Caldwell is a Science & Engineering Librarian at University of California, Santa Cruz. She received her M.S. in Information Science from California State University, San Jose and has a B.S. in Biology from the California State University, Sacramento. Since 2008 she has managed the Library's video game collection that contains over 1,700 console, computer and mobile app video games. Her research interests include information searching strategies of researchers, and research data curation best practices for university researchers and citizen scientists.

**Eric Kaltman**
***Graduate Researcher***
UC Santa Cruz
Computer Science

Eric Kaltman is a PhD candidate in Computer Science at the Expressive Intelligence Studio at UC Santa Cruz. His research focuses on the technical history of computer games and strategies for their long-term preservation. He was a member of the Preserving Virtual Worlds II game preservation project and formerly maintained a blog for Stanford's "How They Got Game" computer game history group. Previously, he has worked on educational research games, networked air quality monitoring systems, and as a digital archivist for Stanford's archival computer game collections, where he oversaw the organization of the Stephen Cabrinety Collection of computer games and hardware, and game developer Steven Meretzky's papers.

**Henry Lowood**
***Curator, History of Science & Technology Collections***
Stanford University

Henry Lowood is curator for history of science & technology collections and for film & media collections at Stanford University. He is also a lecturer in the Thinking Matters Program, the Science and Technology Studies Program and the History and Philosophy of Science Program at Stanford and in the School of Library and Information Science at San Jose State University.

Since 2000, he has led How They Got Game, a research and archival preservation project devoted to the history of digital games and simulations. This project includes Stanford's efforts in the Preserving Virtual Worlds project, funded by the U.S. Library of Congress and the Institute of Museum and Library Services, and the Cabrinety Collection imaging project, funded by the National Institute for Standards and Technology. His most recent book is The Machinima Reader, published by MIT Press and co-edited with Michael Nitsche.

**Noah Wardrip-Fruin**
***Associate Professor***
UC Santa Cruz
Computer Science

Noah Wardrip-Fruin is associate professor of Computational Media at the University of California Santa Cruz, where he co-directs the Expressive Intelligence Studio, one of the world's largest technical research groups focused on games. His research areas include new models of storytelling in games, how games express ideas through play, and the intersections of game studies, software studies, and digital humanities. Wardrip-Fruin also directs the the Playable Media group within UCSC's Digital Arts and New Media MFA program and is an editor of the Software Studies series for MIT Press. His interdisciplinary projects have been supported by US federal funders ranging from the National Science Foundation to the Institute for Museum and Library Services, as well as by corporations such as Microsoft and Google, and have been presented by art and games venues such as the Guggenheim Museum, Whitney Museum of American Art, New Museum of Contemporary Art, IndieCade, and the Independent Games Festival. In 2012 he organized Media Systems, the first joint meeting of the US National Science Foundation, National Endowment for the Arts, and National Endowment for the Humanities. Holding a both a doctorate and MFA from Brown University, he has authored or edited five books on games and digital media for the MIT Press, most recently Expressive Processing: Digital Fictions, Computer Games, and Software Studies (2009).

# Bibliography


Bruemmer, Bruce, and Sheldon Hochheiser. *The High-Technology Company: A Historical Research and Archival Guide.* Charles Babbage Institute, Center for the History of Information Processing, University of Minnesota, 1989.

Elliott, Clark A. *Understanding Progress as Process: Documentation of the History of Post-War Science and Technology in the United States.* Society of American Archivists, 1983.

Haas, Joan K., Helen Willa Samuels, and Barbara Trippel Simmons. *Appraising the Records of Modern Science and Technology: A Guide.* Massachusetts Institute of Technology. Cambridge, MA, 1985.

Khatib, Firas, Seth Cooper, Michael D. Tyka, Kefan Xu, Ilya Makedon, Zoran Popović, David Baker, and Foldit Players. "Algorithm Discovery by Protein Folding Game Players." Proceedings of the National Academy of Sciences 108, no. 47 (2011): 18949–53.

McCoy, Josh, Mike Treanor, Ben Samuel, Aaron A. Reed, Michael Mateas, and Noah Wardrip-Fruin. "Prom Week: Designing Past the Game/Story Dilemma." Proceedings of the 8th International Conference on Foundations of Digital Games, 2013.

McCoy, Joshua, and Michael Mateas. "The Computation of Self in Everyday Life: A Dramaturgical Approach for Socially Competent Agents." In AAAI Spring Symposium: Intelligent Narrative Technologies II, 75–82, 2009.

McCoy, Joshua, Mike Treanor, Ben Samuel, Brandon Robert Tearse, Michael Mateas, and Noah Wardrip-Fruin. "The Prom: An Example of Socially-Oriented Gameplay." In AIIDE, 2010.

McDonough, Jerome P. *Preserving Virtual Worlds: Final Report*. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, 2010.

Von Ahn, Luis, and Laura Dabbish. "Labeling Images with a Computer Game." In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 319–26. ACM, 2004.

Warnow, J. N., A. Needell, S. R. Weart, and J. Wolff. *Study of Preservation of Documents at Department of Energy Laboratories. Final Report.* American Inst. of Physics, New York, January 1, 1982.